A MICROCOMPUTER BASED SHIPBOARD SURFACE-SUBSURFACE CONTACT PLOTTER SYSTEM. PT.2

Antonio Luiz Soares Goncalves

NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

A MICROCOMPUTER BASED SHIPBOARD
SURFACE-SUBSURFACE CONTACT PLOTTER SYSTEM

by

Antonio Luiz Soares Goncalves

and

Javier Enrique De la Cuba Bravo

June 1978

Thesis Advisor: Stephen T. Holl

Approved for public release; distribution unlimited.

U184180

G5469
pt.2
c.1

```
/*****************************************************************************
*
* INPUT$TIME:
*   THIS PROCEDURE IS USED TO ALTER ALL VALUES CONCERNING WITH TIME:
*     - TIME ZONE NUMBER.
*     - SYSTEM CLOCK TIME.
*     - TIME BETWEEN UPDATES OF OWN SHIP POSITIONS.
*
* USAGE:
*   TYPED PROCEDURE.  IF THE TIME BETWEEN UPDATES IS CHANGED (DEFAULT 180 SEC-
*   ONDS) THEN THE NEW VALUE IS RETURNED, OTHERWISE A VALUE OF ZERO IS RETUR-
*   NED.  THE VALUE MUST BE BETWEEN 250 AND 15.
*
*****************************************************************************/
INPUT$TIME: PROCEDURE BYTE PUBLIC;
   DCL ARRAY(3) BYTE,
       VALUE ADDRESS,
       (I, OK, TEMP) BYTE;
   DCL M0 (*) BYTE DATA ('TIME ZONE NUMBER?       (Y/N)  $$'),
       M1 (*) BYTE DATA ('SYSTEM CLOCK VALUE?     (Y/N)  $$'),
       M2 (*) BYTE DATA ('TIME BETWEEN UPDATES?   (Y/N)  $$'),
       M3 (*) BYTE DATA ('ENTER TIME BETWEEN UPDATES AS REQUESTED:$$'),
       M4 (*) BYTE DATA ('SECONDS:  $$'),
       M5 (*) BYTE DATA (' *** BAD FORMAT ***$$');

   OK = 0;
   DO WHILE OK = 0;
      CALL CRT$PRINT$STRING(.TITLE$6);
      CALL SEND$CRLF;
      CALL CRT$PRINT$STRING(.MSG$1);
      CALL SEND$CRLF;
```

```
      CALL CRT$PRINT$STRING(.M0);
      ARRAY(0) = CHECK$YES$NO;
      CALL SEND$SPACE(13);
      CALL CRT$PRINT$STRING(.M1);
      ARRAY(1) = CHECK$YES$NO;
      CALL SEND$CRLF;
      CALL CRT$PRINT$STRING(.M2);
      ARRAY(2) = CHECK$YES$NO;
      CALL SEND$CRLF;
      OK = CHECK$INPUT;
      CALL CLEAR$LOW$SCREEN;
      END;
   DO I = 0 TO 2;
      IF ARRAY(I)
      THEN DO;
           CALL CRT$PRINT$STRING(.TITLE$6);
           CALL SEND$CRLF;
           DO CASE I;
              DO;
                CALL GET$TIME$ZONE(.SYSTEM.NUM$ZONE);
                CALL PRINT$TIME$ZONE(.SYSTEM.NUM$ZONE);
              END;
              DO;
                CALL INITIATE$TIME;
              END;
              DO;
                OK = 0;
                DO WHILE OK = 0;
                   CALL CRT$PRINT$STRING(.M3);
                   CALL SEND$CRLF;
                   VALUE = 0;
```

```
                    DO WHILE (VALUE > 250) OR (VALUE < 15);
                       CALL CRT$PRINT$STRING(.M4);
                       VALUE = GET$ADDRESS(3);
                       IF (VALUE > 250) OR (VALUE < 15)
                       THEN DO;
                            CALL CRT$PRINT$STRING(.M5);
                            CALL SEND$BEL;
                            CALL SEND$CR;
                            CALL SEND$SUB;
                            END;
                       ELSE DO;
                            CALL CRT$WRITE(17H);    /* ERASE TO END OF LINE */
                            CALL SEND$CRLF;
                            END;
                       END;
                    OK = CHECK$INPUT;
                    CALL CLEAR$LOW$SCREEN;
                    END;
              END;
              END;              /* END CASE */
         END;                   /* END THEN DO */
   END;                         /* END DO */
IF ARRAY(2) THEN RETURN LOW(VALUE);
            ELSE RETURN 0;
END INPUT$TIME;


END EXECUTIVE$CMDS;
```

```
CPA$MODULE: DO;
$NOLIST

$INCLUDE (:F1:EXTER.SRC)

$INCLUDE (:F1:EXTER1.SRC)

$LIST

DCL SAFE$RNG (4) BYTE EXTERNAL;
```

```
/*****************************************************************************
*
* CONV$CONTACT$TIME:
*   THIS PROCEDURE IS USED TO CONVERT A GIVEN CONTACT TIME (IN HOURS,
*   MINUTES, AND SECONDS) INTO A FP REPRESENTATION (IN HOURS AND TENTHS
*   OF HOURS).
*
* PARAMETERS:
*   - S. - POINTER TO A MEMORY LOCATION IN WHICH THE TIME IS LOCATED
*          (IN HOURS, MINUTES, AND SECONDS).
*   - T. - POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE REPRESEN-
*          TING THE TIME IS DESIRED TO BE PLACED.
*
*****************************************************************************/
CONV$CONTACT$TIME: PROCEDURE (S, T) PUBLIC;
   DCL (S, T) ADDRESS,
       STRING BASED S (3) BYTE,
       TIME$FLOAT BASED T (4) BYTE,
       TEMP (4) BYTE,
       FP$60 (4) BYTE DATA (00H,00H,70H,42H),
       FP$3600 (4) BYTE DATA (00H,00H,61H,45H);
   TEMP(0) = STRING(0);
   TEMP(1), TEMP(2), TEMP(3) = 00H;
   CALL FLTDS (.TEMP, .TIME$FLOAT);
   TEMP(0) = STRING(1);
   TEMP(1), TEMP(2), TEMP(3) = 00H;
   CALL FLTDS (.TEMP, .TEMP);
   CALL FDIV (.TEMP, .FP$60, .TEMP);
   CALL FADD (.TEMP, .TIME$FLOAT, .TIME$FLOAT);
   TEMP(0) = STRING(2);
   TEMP(1), TEMP(2), TEMP(3) = 00H;
```

```
    CALL FLTDS (.TEMP, .TEMP);
    CALL FDIV (.TEMP, .FP$3600, .TEMP);
    CALL FADD (.TEMP, .TIME$FLOAT, .TIME$FLOAT);
    END CONV$CONTACT$TIME;
```

```
/*****************************************************************************
*
* CPA$TIME$CONV:
*   THIS PROCEDURE IS CALLED BY THE "CPA$CALCULATION" PROCEDURE IN
*   ORDER TO CONVERT A FP REPRESENTATION OF THE CPA TIME TO A STRING OF
*   ASCII CHARACTERS.
*
* PARAMETERS:
*   - T. - POINTER TO A MEMORY LOCATION IN WHICH THE FP REPRESENTATION OF
*          THE CPA TIME IS LOCATED.
*   - S. - POINTER TO A MEMORY LOCATION IN WHICH THE STRING OF CHARACTERS RE-
*          PRESENTING THE CPA TIME IS DESIRED TO BE PLACED.
*
*****************************************************************************/
CPA$TIME$CONV: PROCEDURE (T, S);
   DCL (T, S) ADDRESS,
       FP$60 (4) BYTE DATA (00H,00H,70H,42H),   /* 60.0 */
       CPA$TIME BASED T (4) BYTE,
       STRING BASED S (4) BYTE,
       (HOURS, MINUTES, TEMP) (4) BYTE,
       J BYTE;
   CALL FIXSD (.CPA$TIME, .HOURS);
   DO J = 0 TO 3;
      TEMP(J) = HOURS(J);
      END;
   DO WHILE HOURS(0) >= 24;
      HOURS(0) = HOURS(0) - 24;
      END;
   STRING(0) = HOURS(0) / 10 + 30H;
   STRING(1) = HOURS(0) MOD 10 + 30H;
   CALL FLTDS (.TEMP, .TEMP);
```

```
CALL FSUB (.CPA$TIME, .TEMP, .MINUTES);
CALL FMUL (.MINUTES, .FP$60, .MINUTES);
CALL FIXSD (.MINUTES, .MINUTES);
IF MINUTES(0) >= 60
THEN DO;
     MINUTES(0) = MINUTES(0) - 60;
     HOURS(0) = HOURS(0) + 1;
     IF HOURS(0) >= 24 THEN HOURS(0) = HOURS(0) - 24;
     STRING(0) = HOURS(0) / 10 + 30H;
     STRING(1) = HOURS(0) MOD 10 + 30H;
     END;
STRING(2) = MINUTES(0) / 10 + 30H;
STRING(3) = MINUTES(0) MOD 10 + 30H;
END CPA$TIME$CONV;
```

```
/*****************************************************************************
*
* CONTACT$CRS$SPD:
*   THIS PROCEDURE IS USED TO CALCULATE THE COURSE AND SPEED OF A CONTACT
*   GIVEN ITS LAST KNOWN POSITION AT "CONTACT$POSI" STRUCTURE.
*   THIS PROCEDURE IS CALLED BY THE "CPA$CALCULATION" PROCEDURE.
*
* PARAMETERS:
*   - A. - POINTER TO A MEMORY LOCATION IN WHICH THE RELATIVE COURSE OF A
*          CONTACT IS LOCATED (IN RADIANS).
*   - B. - POINTER TO A MEMORY LOCATION IN WHICH THE RELATIVE SPEED OF  A
*          CONTACT IS LOCATED (IN KNOTS).
*   - S. - POINTER TO A MEMORY LOCATION IN WHICH THE STRING OF CHARACTERS
*          REPRESENTING THE TRUE COURSE AND SPEED OF A CONTACT IS DESIRED
*          TO BE PLACED.
*   - INDEX. - IT IS THE VALUE WHICH GIVES THE LAST KNOWN POSITION  IN
*              THE "CONTACT$POSI" STRUCTURE OF A CONTACT BEING PROCESSED
*              BY THE "CPA$CALCULATION" PROCEDURE.
*
*****************************************************************************/
CONTACT$CRS$SPD: PROCEDURE (A, B, S, INDEX);
   DCL (A, B, S) ADDRESS,
       CRS BASED A (4) BYTE,
       SPD BASED B (4) BYTE,
       STRING BASED S (9) BYTE,
       FP$DEG$TO$RAD (4) BYTE DATA (35H,0FAH,8EH,3CH),   /* 0.0174532925  */
       (TEMP, TEMP1) (4) BYTE,
       (SIN$CRS, COS$CRS) (4) BYTE,
       (X1, X2, XM, Y1, Y2, YM) (4) BYTE,
       (INDEX, I, J) BYTE;
   J = OWN$SHIP$INFO.POINTER;
```

```
   DO I = 0 TO 3;
      TEMP(I) = OWN$SHIP(J).CRS(I);
      TEMP1(I) = OWN$SHIP(J).SPD(I);
      END;
   CALL FMUL (.TEMP, .FP$DEG$TO$RAD, .TEMP);
   CALL COS$SIN (.TEMP, .COS$CRS, .SIN$CRS);
   CALL FMUL (.TEMP1, .SIN$CRS, .X1);
   CALL FMUL (.TEMP1, .COS$CRS, .Y1);
   CALL COS$SIN (.CRS, .COS$CRS, .SIN$CRS);
   CALL FMUL (.SPD, .SIN$CRS, .X2);
   CALL FMUL (.SPD, .COS$CRS, .Y2);
   CALL FADD (.X1, .X2, .XM);
   CALL FADD (.Y1, .Y2, .YM);
   CALL FSQR (.XM, .TEMP);
   CALL FSQR (.YM, .TEMP1);
   CALL FADD (.TEMP, .TEMP1, .TEMP);
   CALL FSQRT (.TEMP, .TEMP);    /*  TRUE SPEED    */
   CALL ARC$TAN (.YM, .XM, .TEMP1);
   CALL FDIV (.TEMP1, .FP$DEG$TO$RAD, .TEMP1);    /*  TRUE COURSE IN DEGREES  */
   DO I = 0 TO 3;
      CONTACT$POSI(INDEX).CRS(I) = TEMP1(I);
      CONTACT$POSI(INDEX).SPD(I) = TEMP(I);
      END;
   J = INDEX/15;
   CONTACT$INFO(J).CRS$FLAG , CONTACT$INFO(J).SPD$FLAG = 0FFH;
   J = FP$FORMAT (.TEMP1, .STRING(0), 3, 1);
   J = FP$FORMAT (.TEMP, .STRING(4), 2, 1);
   END CONTACT$CRS$SPD;
```

```
/*****************************************************************************
*
* CPA$CALCULATION:
*   THIS PROCEDURE IS CALLED BY THE "GET$CPA" PROCEDURE IN ORDER TO
*   CALCULATE THE CPA OF A GIVEN CONTACT.
*   THE "LEAST SQUARE FIT" METHOD IS USED WITH AT MOST 5 POSITIONS OF
*   A GIVEN CONTACT.
*
* PARAMETERS:
*   - INDEX1. - IT INDICATES THE FIRST POSITION AT THE "CONTACT$POSI"
*     STRUCTURE THAT HAS THE INFORMATION ABOUT THE GIVEN CONTACT.
*   - INDEX2. - IT INDICATES THE FIRST POSITION AT THE "CONTACT$POSI"
*     STRUCTURE THAT WILL BE USED IN THE "LEAST SQUARE FIT" COMPUTATION.
*   - COUNT. - IT GIVES THE COUNTING USED TO DETERMINE THE NUMBER OF CONTACT
*     POSITIONS TO BE USED IN THE CALCULATION OF THE CPA.
*   - S. - POINTER TO A MEMORY LOCATION IN WHICH THE STRING OF CHARACTERS
*     REPRESENTING THE CPA INFORMATION IS DESIRED TO BE PLACED.
*
*****************************************************************************/
CPA$CALCULATION: PROCEDURE (INDEX1, INDEX2, COUNT, S);
   DCL S ADDRESS,
       STRING BASED S (23) BYTE,
       CHECK BYTE DATA (04H),              /* CHECK FOR EQUAL */
       CHECK1 BYTE DATA (02H),             /* CHECK FOR GREATER THAN */
       CHECK2 BYTE DATA (05H),             /* CHECK FOR NOT EQUAL */
       FP$DEG$TO$RAD (4) BYTE DATA (35H,0FAH,8EH,3CH), /* 0.0174532925 */
       FP$1 (4) BYTE DATA (00H,00H,80H,3FH),           /* 1.0 */
       FP$60 (4) BYTE DATA (00H,00H,70H,42H),          /* 60.0 */
       PI$OVER2 (4) BYTE DATA (0DBH,0FH,0C9H,3FH),     /* 1.5707963 */
       PI$FLOAT (4) BYTE DATA (0DBH,0FH,49H,40H),      /* 3.141593 */
       PI$3$OVER2 (4) BYTE DATA (0E4H,0CBH,96H,40H),   /* 4.7123889 */
```

```
      MSG0 (*) BYTE DATA
           ('SAME CRS & SPD'),
      MSG1 (*) BYTE DATA
           (' COLLISION'),
      MSG2 (*) BYTE DATA
           ('MOVING AWAY   '),
      REL$XY (5) STRUCTURE
             ( X(4) BYTE,
               Y(4) BYTE),
      COS$BRG (4) BYTE, SIN$BRG (4) BYTE,
      X1 (4) BYTE, Y1 (4) BYTE,
      SLOPE (4) BYTE, Y$CUT (4) BYTE,
      BIG$Y1 (4) BYTE, BIG$Y2 (4) BYTE,
      X$CPA (4) BYTE, Y$CPA (4) BYTE,
      CPA$TIME (4) BYTE, CPA$BRG (4) BYTE, CPA$RNG (4) BYTE,
      TIME (4) BYTE, TIME1 (4) BYTE,
      S0 (4) BYTE, S1 (4) BYTE, S2 (4) BYTE,
      T0 (4) BYTE, T1 (4) BYTE,
      X$SQUARE (4) BYTE, XY$PROD (4) BYTE,
      S1$SQUARE (4) BYTE, ST$PROD (4) BYTE,
      NUMERATOR (4) BYTE, DENOMINATOR (4) BYTE,
      (REL$CRS, REL$SPD, TEMP$RAD) (4) BYTE,
      (INDEX1, INDEX2, COUNT, TEMP, TEMP1, I, J, FLAG, FLAG1, LAST$POINTER) BYTE;

   /* LAST$POINTER WILL POINT TO THE LAST POSITION IN CONTACT$POSI  */
TEMP = INDEX1 / 15;
LAST$POINTER = CONTACT$INFO(TEMP).POINTER;
TEMP = INDEX2;
DO I = 0 TO 3;
   S0(I), S1(I), S2(I), T0(I), T1(I) = 00H;
   END;
```

```
            /* COMPUTE PARAMETERS FOR LEAST SQUARE FIT */
   S0(0) = COUNT + 1;
   CALL FLTDS (.S0, .S0);
   DO I = 0 TO COUNT;
      IF TEMP > INDEX1 + 14 THEN TEMP = INDEX1;
      CALL FMUL (.CONTACT$POSI(TEMP).BRG, .FP$DEG$TO$RAD, .TEMP$RAD);
      CALL COS$SIN (.TEMP$RAD, .COS$BRG, .SIN$BRG);
      CALL FMUL (.CONTACT$POSI(TEMP).RNG, .SIN$BRG, .REL$XY(I).X);
      CALL FMUL (.CONTACT$POSI(TEMP).RNG, .COS$BRG, .REL$XY(I).Y);
      CALL FADD (.REL$XY(I).X, .S1, .S1);
      CALL FSQR (.REL$XY(I).X, .X$SQUARE);
      CALL FADD (.X$SQUARE, .S2, .S2);
      CALL FADD (.REL$XY(I).Y, .T0, .T0);
      CALL FMUL (.REL$XY(I).Y, .REL$XY(I).X, .XY$PROD);
      CALL FADD (.XY$PROD, .T1, .T1);
      TEMP = TEMP + 1;
      END;
   CALL FMUL (.S0, .S2, .DENOMINATOR);
   CALL FSQR (.S1, .S1$SQUARE);
   CALL FSUB (.DENOMINATOR, .S1$SQUARE, .DENOMINATOR);
   CALL FMUL (.S0, .T1, .NUMERATOR);
   CALL FMUL (.S1, .T0, .ST$PROD);
   CALL FSUB (.NUMERATOR, .ST$PROD, .NUMERATOR);
   FLAG = FZTST (.DENOMINATOR, .CHECK);
   FLAG1 = FZTST (.NUMERATOR, .CHECK);
   IF FLAG AND FLAG1 THEN I = 0;              /* SLOPE = 0/0 */
   IF FLAG AND (NOT FLAG1) THEN I = 1;        /* SLOPE = X/0 */
   IF (NOT FLAG) AND FLAG1 THEN I = 2;        /* SLOPE = 0 */
   IF (NOT FLAG) AND (NOT FLAG1) THEN I = 3; /* SLOPE <> 0 */
   TEMP = LAST$POINTER;
   IF TEMP = INDEX1
```

```
      THEN TEMP1 = TEMP + 14;
      ELSE TEMP1 = LAST$POINTER - 1;
   IF (I = 0) AND (FLAG1 := FCMPR(.CONTACT$POSI(TEMP).RNG,
                                  .CONTACT$POSI(TEMP1).RNG, .CHECK2))
      THEN I = 1;
   IF (I = 1) OR (I = 3)
   THEN DO;
        IF FCMPR(.CONTACT$POSI(TEMP).BRG,.CONTACT$POSI(TEMP1).BRG,.CHECK) AND
           FCMPR(.CONTACT$POSI(TEMP).RNG,.CONTACT$POSI(TEMP1).RNG,.CHECK)
        THEN I = 0;
        END;
   DO CASE I;
      DO;           /* CONTACT ON SAME CRS & SPD OF OWN SHIP */
         TEMP = LAST$POINTER;
         TEMP1 = OWN$SHIP$INFO.POINTER;
         DO J = 0 TO 3;
            CONTACT$POSI(TEMP).CRS(J) = OWN$SHIP(TEMP1).CRS(J);
            CONTACT$POSI(TEMP).SPD(J) = OWN$SHIP(TEMP1).SPD(J);
            END;
         TEMP1 = FP$FORMAT(.CONTACT$POSI(TEMP).CRS, .STRING(0), 3, 1);
         TEMP1 = FP$FORMAT(.CONTACT$POSI(TEMP).SPD, .STRING(4), 2, 1);
         TEMP1 = INDEX1 / 15;
         CONTACT$INFO(TEMP1).CRS$FLAG = 0FFH;
         CONTACT$INFO(TEMP1).SPD$FLAG = 0FFH;
         DO J = 7 TO LAST(STRING);
            STRING(J) = MSG0(J - 7);
            END;
      END;
      DO;           /* CONTACT RELATIVE COURSE = 000 OR 180 */
         TEMP = LAST$POINTER;
         IF TEMP = INDEX1
```

```
   THEN TEMP1 = TEMP + 14;
   ELSE TEMP1 = LAST$POINTER - 1;
FLAG = FCMPR(.CONTACT$POSI(TEMP1).RNG,.CONTACT$POSI(TEMP).RNG,.CHECK1);
CALL FMUL (.CONTACT$POSI(INDEX2).BRG, .FP$DEG$TO$RAD, .TEMP$RAD);
CALL COS$SIN(.TEMP$RAD, .COS$BRG,.SIN$BRG);
CALL FMUL(.CONTACT$POSI(INDEX2).RNG,.SIN$BRG,.X$CPA);
IF X$CPA(3) >= 80H
   THEN X$CPA(3) = X$CPA(3) XOR 080H;
      /* CONVERT TIME TO FP */
TEMP = LAST$POINTER;
CALL CONV$CONTACT$TIME(.CONTACT$POSI(INDEX2).TIME, .TIME);
IF CONTACT$POSI(INDEX2).TIME(0) >
   CONTACT$POSI(TEMP).TIME(0)
THEN DO;
   CONTACT$POSI(TEMP).TIME(0) = CONTACT$POSI(TEMP).TIME(0) + 24;
   CALL CONV$CONTACT$TIME(.CONTACT$POSI(TEMP).TIME,.TIME1);
   CONTACT$POSI(TEMP).TIME(0) = CONTACT$POSI(TEMP).TIME(0) - 24;
   END;
ELSE CALL CONV$CONTACT$TIME(.CONTACT$POSI(TEMP).TIME,.TIME1);
      /* COMPUTE RELATIVE COURSE */
IF (FLAG1 := FCMPR(.REL$XY(COUNT).Y, .REL$XY(0).Y, .CHECK1))
THEN DO;
    DO J = 0 TO 3;
       REL$CRS(J) = 00H;
       END;
    END;
ELSE DO;
    DO J = 0 TO 3;
       REL$CRS(J) = PI$FLOAT(J);
       END;
    END;
```

```
        /* COMPUTE RELATIVE SPEED */
CALL FSUB (.TIME1, .TIME, .TIME1);
CALL FSUB (.REL$XY(COUNT).Y, .REL$XY(0).Y, .Y1);
IF Y1(3) >= 80H THEN Y1(3) = Y1(3) XOR 80H;
CALL FDIV (.Y1, .TIME1, .REL$SPD);
        /* COMPUTE TRUE COURSE AND SPEED */
TEMP = LAST$POINTER;
CALL CONTACT$CRS$SPD (.REL$CRS, .REL$SPD, .STRING, TEMP);
IF FLAG
THEN DO;          /* CONTACT CLOSING */
          /* COMPUTE CPA TIME */
   CALL FMUL (.CONTACT$POSI(INDEX2).RNG, .COS$BRG, .Y$CPA);
   CALL FDIV (.Y$CPA, .REL$SPD, .CPA$TIME);
   IF CPA$TIME(3) >= 080H
      THEN CPA$TIME(3) = CPA$TIME(3) XOR 080H;
   CALL FADD(.CPA$TIME, .TIME, .CPA$TIME);
   CALL CPA$TIME$CONV (.CPA$TIME, .STRING(7));
          /* CHECK FOR COLLISION */
   IF (FLAG1 := FCMPR (.SAFE$RNG, .X$CPA, .CHECK1))
   THEN DO;
      DO J = 11 TO LAST(STRING);
         STRING(J) = MSG1(J - 11);
         END;
      RETURN;
      END;
          /* COMPUTE CPA BEARING */
   CALL FMUL (.CONTACT$POSI(INDEX2).BRG, .FP$DEG$TO$RAD, .TEMP$RAD);
   IF (FLAG1 := FCMPR(.PI$FLOAT,.TEMP$RAD, .CHECK1))
   THEN DO;
      STRING(11) = '0';
      STRING(12) = '9';
```

```
            STRING(13) = '0';
            STRING(14) = '0';
            END;
         ELSE DO;
            STRING(11) = '2';
            STRING(12) = '7';
            STRING(13) = '0';
            STRING(14) = '0';
            END;  '
                /* COMPUTE CPA RANGE */
         CALL RANGE$FORMAT (.X$CPA, .STRING(15));
         END;
      ELSE DO;  /* CONTACT MOVING AWAY */
         DO J = 7 TO LAST(STRING);
            STRING(J) = MSG2(J - 7);
            END;
         END;
   END;
   DO;            /* CONTACT RELATIVE COURSE = 090 OR 270 */
      TEMP = LAST$POINTER;
      IF TEMP = INDEX1
         THEN TEMP1 = TEMP + 14;
         ELSE TEMP1 = LAST$POINTER - 1;
      FLAG = FCMPR(.CONTACT$POSI(TEMP1).RNG, .CONTACT$POSI(TEMP).RNG,
                   .CHECK1);
      CALL FMUL (.CONTACT$POSI(INDEX2).BRG, .FP$DEG$TO$RAD, .TEMP$RAD);
      CALL COS$SIN(.TEMP$RAD, .COS$BRG,.SIN$BRG);
      CALL FMUL(.CONTACT$POSI(INDEX2).RNG,.COS$BRG,.Y$CPA);
      IF Y$CPA(3) >= 080H
         THEN Y$CPA(3) = Y$CPA(3) XOR 080H;
                     /* CONVERT TIME TO FP */
```

```
TEMP = LAST$POINTER;
CALL CONV$CONTACT$TIME(.CONTACT$POSI(INDEX2).TIME,.TIME);
IF CONTACT$POSI(INDEX2).TIME(0) >
   CONTACT$POSI(TEMP).TIME(0)
THEN DO;
   CONTACT$POSI(TEMP).TIME(0) = CONTACT$POSI(TEMP).TIME(0) + 24;
   CALL CONV$CONTACT$TIME(.CONTACT$POSI(TEMP).TIME,.TIME1);
   CONTACT$POSI(TEMP).TIME(0) = CONTACT$POSI(TEMP).TIME(0) - 24;
   END;
ELSE CALL CONV$CONTACT$TIME(.CONTACT$POSI(TEMP).TIME,.TIME1);
     /* COMPUTE RELATIVE COURSE */
IF(FLAG1:= FCMPR (.REL$XY(COUNT).X, .REL$XY(0).X, .CHECK1))
THEN DO;
     DO J = 0 TO 3;
        REL$CRS(J) = PI$OVER2(J);
        END;
     END;
ELSE DO;
     DO J = 0 TO 3;
        REL$CRS(J) = PI$3$OVER2(J);
        END;
     END;
     /* COMPUTE RELATIVE SPEED */
CALL FSUB (.TIME1, .TIME, .TIME1);
CALL FSUB (.REL$XY(COUNT).X,.REL$XY(0).X, .X1);
IF X1(3) >= 80H THEN X1(3) = X1(3) XOR 80H;
CALL FDIV(.X1, .TIME1, .REL$SPD);
        /* COMPUTE TRUE COURSE AND SPEED */
TEMP = LAST$POINTER;
CALL CONTACT$CRS$SPD (.REL$CRS, .REL$SPD, .STRING, TEMP);
IF FLAG
```

```
THEN DO;          /* CONTACT CLOSING */
                  /* COMPUTE CPA TIME */
   CALL FMUL(.CONTACT$POSI(INDEX2).RNG, .SIN$BRG, .X$CPA);
   CALL FDIV(.X$CPA, .REL$SPD, .CPA$TIME);
   IF CPA$TIME(3) >= 080H
      THEN CPA$TIME(3) = CPA$TIME(3) XOR 080H;
   CALL FADD(.CPA$TIME, .TIME, .CPA$TIME);
   CALL CPA$TIME$CONV (.CPA$TIME, .STRING(7));
            /* CHECK FOR COLLISION */
   IF (FLAG1 := FCMPR (.SAFE$RNG, .Y$CPA, .CHECK1))
   THEN DO;
        DO J = 11 TO LAST(STRING);
           STRING(J) = MSG1(J - 11);
           END;
        RETURN;
        END;
           /* COMPUTE CPA BEARING */
   CALL FMUL(.CONTACT$POSI(INDEX2).BRG, .FP$DEG$TO$RAD, .TEMP$RAD);
   IF (FLAG := FCMPR(.PI$3$OVER2,.TEMP$RAD, .CHECK1))
   AND (FLAG:= FCMPR(.TEMP$RAD, .PI$OVER2,.CHECK1))
   THEN DO;
      STRING(11) = '1';
      STRING(12) = '8';
      STRING(13) = '0';
      STRING(14) = '0';
      END;
   ELSE DO;
      STRING(11) = '0';
      STRING(12) = '0';
      STRING(13) = '0';
      STRING(14) = '0';
```

```
            END;
              /* COMPUTE CPA RANGE */
         CALL RANGE$FORMAT (.Y$CPA, .STRING(15));
         END;
      ELSE DO; /* CONTACT MOVING AWAY */
         DO J = 7 TO LAST(STRING);
            STRING(J) = MSG2(J - 7);
            END;
         END;
   END;
   DO;          /* RELATIVE MOVE - GENERAL CASE */
      TEMP = LAST$POINTER;
      IF TEMP = INDEX1
          THEN TEMP1 = TEMP + 14;
          ELSE TEMP1 = LAST$POINTER - 1;
      FLAG = FCMPR(.CONTACT$POSI(TEMP1).RNG,.CONTACT$POSI(TEMP).RNG,
                        .CHECK1);
   /* PERFORM LEAST SQUARE FIT FOR LAST POSITIONS */
                         /* COMPUTE SLOPE */
      CALL FDIV (.NUMERATOR, .DENOMINATOR, .SLOPE);
                          /* COMPUTE CUT AT Y AXIS */
      CALL FMUL (.S2, .T0, .NUMERATOR);
      CALL FMUL (.S1, .T1, .ST$PROD);
      CALL FSUB (.NUMERATOR, .ST$PROD, .NUMERATOR);
      CALL FDIV (.NUMERATOR, .DENOMINATOR, .Y$CUT);
         /* COMPUTE RELATIVE COURSE */
         /* PREPARE TO COMPUTE RELATIVE SPEED */
      CALL FMUL(.SLOPE, .REL$XY(0).X, .BIG$Y1);
      CALL FADD(.BIG$Y1, .Y$CUT, .BIG$Y1);
      CALL FMUL(.SLOPE, .REL$XY(COUNT).X, .BIG$Y2);
      CALL FADD(.BIG$Y2, .Y$CUT, .BIG$Y2);
```

```
CALL FSUB(.BIG$Y2, .BIG$Y1, .NUMERATOR);
CALL FSUB(.REL$XY(COUNT).X, .REL$XY(0).X, .DENOMINATOR);
CALL ARC$TAN(.NUMERATOR, .DENOMINATOR, .REL$CRS);
CALL FSQR(.NUMERATOR, .NUMERATOR);
CALL FSQR(.DENOMINATOR, .DENOMINATOR);
CALL FADD(.NUMERATOR, .DENOMINATOR, .NUMERATOR);
CALL FSQRT(.NUMERATOR, .NUMERATOR);
                    /* CONVERT TIME TO FP */
TEMP = LAST$POINTER;
CALL CONV$CONTACT$TIME(.CONTACT$POSI(INDEX2).TIME, .TIME);
IF CONTACT$POSI(INDEX2).TIME(0) >
CONTACT$POSI(TEMP).TIME(0)
THEN DO;
   CONTACT$POSI(TEMP).TIME(0) = CONTACT$POSI(TEMP).TIME(0) + 24;
   CALL CONV$CONTACT$TIME(.CONTACT$POSI(TEMP).TIME, .TIME1);
   CONTACT$POSI(TEMP).TIME(0) = CONTACT$POSI(TEMP).TIME(0) - 24;
   END;
ELSE DO;
     CALL CONV$CONTACT$TIME(.CONTACT$POSI(TEMP).TIME, .TIME1);
     END;
CALL FSUB(.TIME1, .TIME, .TIME1);
                  /* COMPUTE RELATIVE SPEED */
CALL FDIV(.NUMERATOR, .TIME1, .REL$SPD);
          /* COMPUTE TRUE COURSE AND SPEED */
TEMP = LAST$POINTER;
CALL CONTACT$CRS$SPD (.REL$CRS, .REL$SPD, .STRING, TEMP);
IF FLAG `
THEN DO;          /* CONTACT CLOSING */
                      /* COMPUTE CPA */
   CALL FMUL(.SLOPE, .REL$XY(0).X, .NUMERATOR);
   CALL FSUB(.NUMERATOR, .BIG$Y1, .NUMERATOR);
```

```
CALL FMUL(.SLOPE, .NUMERATOR, .NUMERATOR);
CALL FSQR(.SLOPE, .DENOMINATOR);
CALL FADD(.DENOMINATOR, .FP$1, .DENOMINATOR);
                /* COMPUTE X$CPA */
CALL FDIV(.NUMERATOR, .DENOMINATOR, .X$CPA);
                    /* COMPUTE Y$CPA */
CALL FMUL(.SLOPE, .REL$XY(0).X, .NUMERATOR);
CALL FSUB(.BIG$Y1, .NUMERATOR, .NUMERATOR);
CALL FDIV(.NUMERATOR, .DENOMINATOR, .Y$CPA);
                /* COMPUTE CPA TIME */
CALL FSUB(.X$CPA, .REL$XY(0).X, .NUMERATOR);
CALL FSQR(.NUMERATOR, .NUMERATOR);
CALL FSUB(.Y$CPA, .BIG$Y1, .DENOMINATOR);
CALL FSQR(.DENOMINATOR, .DENOMINATOR);
CALL FADD(.NUMERATOR, .DENOMINATOR, .NUMERATOR );
CALL FSQRT(.NUMERATOR, .NUMERATOR);
CALL FDIV(.NUMERATOR, .REL$SPD, .CPA$TIME);
CALL FADD(.TIME, .CPA$TIME, .CPA$TIME);
CALL CPA$TIME$CONV (.CPA$TIME, .STRING(7));
                /* COMPUTE CPA RANGE */
CALL FSQR(.X$CPA, .NUMERATOR);
CALL FSQR(.Y$CPA, .DENOMINATOR);
CALL FADD(.NUMERATOR, .DENOMINATOR, .NUMERATOR);
CALL FSQRT(.NUMERATOR, .CPA$RNG);
                /* CHECK FOR COLLISION */
IF (FLAG1 := FCMPR (.SAFE$RNG, .CPA$RNG, .CHECK1))
   THEN DO;
        DO J = 11 TO LAST(STRING);
           STRING(J) = MSG1(J - 11);
           END;
        RETURN;
```

```
                    END;
                     /* COMPUTE CPA BEARING */
            CALL ARC$TAN(.Y$CPA, .X$CPA, .CPA$BRG);
            CALL CONV$RAD$MIN(.CPA$BRG, .CPA$BRG);
            CALL FDIV(.CPA$BRG, .FP$60, .CPA$BRG);
            TEMP = FP$FORMAT (.CPA$BRG, .STRING(11), 3, 1);
            CALL RANGE$FORMAT(.CPA$RNG, .STRING(15));
            END;
         ELSE DO;
                          /* CONTACT MOVING AWAY */
            DO J = 7 TO LAST(STRING);
               STRING(J) = MSG2(J - 7);
               END;
            END;
      END;

   END;     /* END CASE */
END CPA$CALCULATION;
```

```
/*****************************************************************************
*
* GET$CPA:
*   THIS PROCEDURE IS USED TO FIND THE CPA INFORMATION ABOUT A CONTACT.
*
* PARAMETERS:
*   - INDEX. - INDICATES THE RELATIVE POSITION OF THE CONTACT IN THE
*     CONTACT$INFO STRUCTURE.
*   - A. - POINTER TO A MEMORY LOCATION IN WHICH THE STRING OF CHARACTERS
*     REPRESENTING THE CPA INFORMATION IS DESIRED TO BE PLACED.
*
* USAGE:
*   TYPED PROCEDURE.  RETURNS A VALUE OF 0 IF NO CPA CAN BE DETERMINED AT
*   THE MOMENT.  OTHERWISE, A VALUE OF 1 IS RETURNED.
*
*****************************************************************************/
GET$CPA: PROCEDURE (INDEX, A) BYTE PUBLIC;
   DCL A ADDRESS,
       STRING BASED A (23) BYTE,
       (INDEX, INDEX1, COUNT, TEMP, P1, P2, I) BYTE;
   DO I = 0 TO LAST(STRING);
      STRING(I) = 020H;          /*  PUT BLANKS IN WORK BUFFER */
      END;
   INDEX1 = 15*INDEX;
   IF (CONTACT$INFO(INDEX).POINTER < INDEX1 + 1) AND
      (NOT CONTACT$INFO(INDEX).FLAG)
   THEN RETURN 0;
   P1 = CONTACT$INFO(INDEX).POINTER;
   P2 = CONTACT$INFO(INDEX).OS$POINTER;
   IF CONTACT$INFO(INDEX).FLAG
   THEN DO;
```

```
        IF P2 = 0FFH
        THEN DO;
             COUNT = 4;
             IF P1 < INDEX1 + 4
                THEN TEMP = P1 + 11;
                ELSE TEMP = P1 - 4;
             END;
        ELSE DO;
             COUNT = 0;
             TEMP = P2 + 1;
             IF TEMP = (INDEX1 + 15)
                THEN TEMP = INDEX1;
             DO WHILE TEMP <> P1;
                IF TEMP = (INDEX1 + 15)
                   THEN TEMP = INDEX1;
                   ELSE TEMP = TEMP + 1;
                COUNT = COUNT + 1;
                END;
             IF COUNT = 0
                THEN RETURN 0;
             TEMP = P2 + 1;
             IF TEMP = (INDEX1 + 15)
                THEN TEMP = INDEX1;
             END;
        END;
   ELSE DO;
        IF P2 = 0FFH
        THEN DO;
             COUNT = P1 - INDEX1;
             IF COUNT > 4 THEN COUNT = 4;
             TEMP = P1 - COUNT;
```

```
            END;
        ELSE DO;
            IF P1 = P2 THEN RETURN 0;
            COUNT = P1 - P2 - 1;
            TEMP = P2 + 1;
            IF COUNT = 0 THEN RETURN 0;
            END;
        END;
    CALL CPA$CALCULATION(INDEX1, TEMP, COUNT, .STRING);
    RETURN 1;
    END GET$CPA;
```



```
END CPA$MODULE;
```

DISPLAY$CMDS

```
DISPLAY$CMDS: DO;

$NOLIST

$INCLUDE (:F1:EXTER.SRC)

$INCLUDE (:F1:EXTER1.SRC)

$LIST

GET$CPA:
   PROCEDURE (A,B) BYTE EXTERNAL;
   DCL A BYTE, B ADDRESS; END;

CONV$CONTACT$TIME:
   PROCEDURE (A, B) EXTERNAL;
   DCL (A, B) ADDRESS; END;


DCL SAFE$RNG (4) BYTE EXTERNAL;

DCL
    TITLE$0 (*) BYTE DATA
            ('
    TITLE$1 (*) BYTE DATA
            ('
    TITLE$2 (*) BYTE DATA
            ('
    TITLE$3 (*) BYTE DATA
```

```
COORDINATE GRID ORIGIN$$'),

    GRAPHICS SCALE$$'),

 OWN SHIP INFORMATION$$'),
```

```
            ('                                    CONTACT INFORMATION$$'),
    TITLE$4 (*) BYTE DATA
            ('                                     SYSTEM INFORMATION. $$'),
    TITLE$5 (*) BYTE DATA
            ('                                  CURRENT SAFE C. P. A.  RANGE $$'),
    TITLE$6 (*) BYTE DATA
            ('                                        WIND INFORMATION$$'),
    TITLE$7 (*) BYTE DATA
            ('                                 CURRENT TIME BETWEEN UPDATES. $$');

DCL
    MSG$00 (*) BYTE DATA ('POSITIONAL DATA:$$'),
    MSG$01 (*) BYTE DATA ('TACTICAL DATA AT $$'),
    MSG$02 (*) BYTE DATA ('C. P. A.  DATA:$$'),
    MSG$03 (*) BYTE DATA ('GENERAL DATA:$$');

DCL
    PLUS$SIGN  LIT '02BH',
    MINUS$SIGN LIT '02DH',
    COLON LIT '03AH',
    POINT LIT '02EH',
    BLANK LIT '020H';
```

```
/*****************************************************************************
*
* CONV$LAT$LONG:
*   THIS PROCEDURE IS USED TO CONVERT GIVEN 'X,Y' COORDINATES, INTO LATITUDE
*   AND LONGITUDE VALUES.
*
* PARAMETERS:
*   - A. - POINTER TO A MEMORY LOCATION IN WHICH THE FP REPRESENTATION OF 'X'
*     IS LOCATED.
*   - B. - POINTER TO A MEMORY LOCATION IN WHICH THE FP REPRESENTATION OF 'Y'
*     IS LOCATED.
*   - C. - POINTER TO A MEMORY LOCATION IN WHICH THE VALUE OF THE LATITUDE, IN
*     MINUTES, IS DESIRED TO BE PLACED.
*   - D. - POINTER TO A MEMORY LOCATION IN WHICH THE VALUE OF THE LONGITUDE, IN
*     MINUTES, IS DESIRED TO BE PLACED.
*
*****************************************************************************/
CONV$LAT$LONG: PROCEDURE (A,B,C,D) PUBLIC;
  DCL (A,B,C,D) ADDRESS,
      X BASED A (4) BYTE,
      Y BASED B (4) BYTE,
      LAT BASED C (4) BYTE,
      LONG BASED D (4) BYTE,
      MEAN$LAT (4) BYTE,
      COS$MEAN$LAT (4) BYTE,
      SIN$MEAN$LAT (4) BYTE,
      FP$2 (4) BYTE DATA (00H, 00H, 00H, 40H);
  CALL FADD(.Y, .SYSTEM.LAT, .LAT);
  CALL FADD(.SYSTEM.LAT, .LAT, .MEAN$LAT);
  CALL FDIV(.MEAN$LAT, .FP$2, .MEAN$LAT);
  CALL CONV$MIN$RAD(.MEAN$LAT, .MEAN$LAT);
```

```
CALL COS$SIN(.MEAN$LAT, .COS$MEAN$LAT, .SIN$MEAN$LAT);
CALL FDIV(.X, .COS$MEAN$LAT, .LONG);
CALL FADD(.LONG, .SYSTEM.LONG, .LONG);
END CONV$LAT$LONG;
```

```
/*****************************************************************************
*
* DISPLAY$DESIG:
*   THIS PROCEDURE IS USED TO DISPLAY THE DESIG CHARACTERS.
*
* PARAMETERS:
*   - DESIG.- ADDRESS VALUE REPRESENTING THE 'HASHED' VALUE OF THE DESIGNATION
*     DESIRED TO BE DISPLAYED.
*
*****************************************************************************/
DISPLAY$DESIG: PROCEDURE(DESIG) PUBLIC;
   DCL DESIG ADDRESS,
       CHAR(4) BYTE;
   DCL D (10) BYTE DATA ('DESIG:  $$');
   CALL CRT$PRINT$STRING(.D);
   CALL DE$HASH(DESIG, .CHAR);
   CHAR(2), CHAR(3) = '$';
   CALL CRT$PRINT$STRING(.CHAR);
   END DISPLAY$DESIG;
```

```
/*****************************************************************************
*
* DISPLAY$TYPE:
*   THIS PROCEDURE IS USED TO DISPLAY THE TYPE OF A CONTACT.
*
* PARAMETERS:
*   - A. - POINTER TO A BYTE VALUE REPRESENTING THE TYPE TO BE DISPLAYED.
*
*****************************************************************************/
DISPLAY$TYPE: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       TYPE BASED A BYTE;
   DCL S  (*) BYTE DATA ('SURFACE    $$'),
       SS (*) BYTE DATA ('SUB-SURFACE$$'),
       T (10) BYTE DATA ('TYPE:   $$');

   CALL CRT$PRINT$STRING(.T);
   IF TYPE = 0
      THEN CALL CRT$PRINT$STRING(.S);
      ELSE CALL CRT$PRINT$STRING(.SS);
   END DISPLAY$TYPE;
```

```
/*****************************************************************************
*
* DISPLAY$CLASS:
*   THIS PROCEDURE IS USED TO DISPLAY THE CLASS OF A CONTACT.
*
* PARAMETERS:
*   - A. - POINTER TO A MEMORY LOCATION IN WHICH THE CLASS TO BE DISPLAYED IS
*     LOCATED.
*
*****************************************************************************/
DISPLAY$CLASS: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       CLASS BASED A BYTE;
   DCL FRI (*) BYTE DATA (' FRIEND$$'),
       HOS (*) BYTE DATA ('HOSTILE$$'),
       UNK (*) BYTE DATA ('UNKNOWN$$'),
       C (10) BYTE DATA ('CLASS:  $$');

   CALL CRT$PRINT$STRING(.C);
   DO CASE CLASS;
      CALL CRT$PRINT$STRING(.FRI);
      CALL CRT$PRINT$STRING(.HOS);
      CALL CRT$PRINT$STRING(.UNK);
      END;
   END DISPLAY$CLASS;
```

```
/*****************************************************************************
*
* DISPLAY$LAT$LONG:
*   THIS PROCEDURE IS USED TO DISPLAY THE VALUES OF LATITUDE AND LONGITUDE.
*
* PARAMETERS:
*   - KIND. - DENOTES LATITUDE IF 0, OTHERWISE DENOTES LONGITUDE.
*   - A. - POINTER TO THE FP REPRESENTATION OF LAT/LONG.
*
*****************************************************************************/
DISPLAY$LAT$LONG: PROCEDURE (KIND, A) PUBLIC;
   DCL A ADDRESS,
       VALUE BASED A (4) BYTE,
       CHAR (9) BYTE,
       (KIND, I) BYTE;
   DCL LAT (14) BYTE DATA ('LATITUDE:   $$'),
       LONG (14) BYTE DATA ('LONGITUDE:  $$'),
       NORTH (8) BYTE DATA (' NORTH$$'),
       SOUTH (8) BYTE DATA (' SOUTH$$'),
       EAST  (8) BYTE DATA (' EAST $$'),
       WEST  (8) BYTE DATA (' WEST $$');

   IF KIND = 0
   THEN DO;
        CALL CRT$PRINT$STRING(.LAT);
        CALL LAT$LONG$FORMAT(A, .CHAR, 0);
        END;
   ELSE DO;
        CALL CRT$PRINT$STRING(.LONG);
        CALL LAT$LONG$FORMAT(A, .CHAR, 1);
        END;
```

```
DO I = 0 TO 5;
   IF I = 3 THEN CALL CRT$WRITE(COLON);
   IF I = 5 THEN CALL CRT$WRITE(POINT);
   CALL CRT$WRITE(CHAR(I));
   END;
IF KIND = 0
THEN DO;
     IF CHAR(6) = 'N'
        THEN CALL CRT$PRINT$STRING(.NORTH);
        ELSE CALL CRT$PRINT$STRING(.SOUTH);
     END;
ELSE DO;
     IF CHAR(6) = 'E'
        THEN CALL CRT$PRINT$STRING(.EAST);
        ELSE CALL CRT$PRINT$STRING(.WEST);
     END;
END DISPLAY$LAT$LONG;
```

```
/*****************************************************************************
*
* DISPLAY$XY:
*   THIS PROCEDURE IS USED TO DISPLAY VALUES OF 'X,Y' COORDINATES.
*
* PARAMETERS:
*   - TYPE. - IF 0 'X' WILL BE DISPLAYED, OTHERWISE 'Y' WILL BE DISPLAYED.
*   - A. - POINTER TO A FP REPRESENTATION OF THE X/Y VALUE.
*
*****************************************************************************/
DISPLAY$XY: PROCEDURE (TYPE, A) PUBLIC;
   DCL A ADDRESS,
       CHAR (14) BYTE,
       (TYPE, TEMP, I) BYTE;
   DCL X (8) BYTE DATA ('X:     $'),
       Y (8) BYTE DATA ('Y:     $');

   IF TYPE = 0
      THEN CALL CRT$PRINT$STRING(.X);
      ELSE CALL CRT$PRINT$STRING(.Y);
   TEMP = FP$FORMAT(A, .CHAR, 10, 2);
   IF TEMP = 0
      THEN CALL CRT$WRITE(PLUS$SIGN);
      ELSE CALL CRT$WRITE(MINUS$SIGN);
   DO I = 0 TO 11;
      IF I = 10 THEN CALL CRT$WRITE(POINT);
      CALL CRT$WRITE(CHAR(I));
      END;
   END DISPLAY$XY;
```

```
/*****************************************************************************
*
* DISPLAY$CRS$BRG:
*   THIS PROCEDURE IS USED TO DISPLAY THE VALUES OF COURSE AND BEARING.
*
* PARAMETERS:
*   - KIND. - IF 0 THE VALUE OF COURSE WILL BE DISPLAYED, OTHERWISE THE VALUE
*     OF BEARING WILL BE DISPLAYED.
*   - A. - POINTER TO THE FP REPRESENTATION OF COURSE/BEARING.
*
*****************************************************************************/
DISPLAY$CRS$BRG: PROCEDURE (KIND, A) PUBLIC;
   DCL A ADDRESS,
       CHAR (6) BYTE,
       (I, TEMP, KIND) BYTE;
   DCL CRS (12) BYTE DATA ('COURSE:    $$'),
       BRG (12) BYTE DATA ('BEARING:   $$');

   IF KIND = 0
      THEN CALL CRT$PRINT$STRING(.CRS);
      ELSE CALL CRT$PRINT$STRING(.BRG);
   TEMP = FP$FORMAT(A, .CHAR, 3, 1);
   DO I = 0 TO 3;
      IF I = 3 THEN CALL CRT$WRITE(POINT);
      CALL CRT$WRITE(CHAR(I));
      END;
   END DISPLAY$CRS$BRG;
```

```
/*****************************************************************************
*
* DISPLAY$SPD:
*   THIS PROCEDURE IS USED TO DISPLAY THE VALUE OF SPEED.
*
* PARAMETERS:
*   - A. - POINTER TO THE FP REPRESENTATION OF THE VALUE OF SPEED.
*
*****************************************************************************/
DISPLAY$SPD: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       CHAR (5) BYTE,
       (TEMP, I) BYTE;
   DCL SPD (12) BYTE DATA ('SPEED:     $$');

   CALL CRT$PRINT$STRING(.SPD);
   TEMP = FP$FORMAT(A, .CHAR, 2, 1);
   DO I = 0 TO 2;
      IF I = 2 THEN CALL CRT$WRITE(POINT);
      CALL CRT$WRITE(CHAR(I));
      END;
   END DISPLAY$SPD;
```

```
/*****************************************************************************
*
* DISPLAY$RANGE:
*    THIS PROCEDURE IS USD TO DISPLAY THE VALUE OF RANGE.
*
* PARAMETERS:
*    - A. - POINTER TO THE FP REPRESENTATION OF RANGE.
*
*****************************************************************************/
DISPLAY$RANGE: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       CHAR (8) BYTE,
       I BYTE;
   DCL MLS (8) BYTE DATA (' MILES$$'),
       YDS (8) BYTE DATA (' YARDS$$'),
       RNG (12) BYTE DATA ('RANGE:    $$');

   CALL CRT$PRINT$STRING(.RNG);
   CALL RANGE$FORMAT(A, .CHAR);
   DO I = 0 TO 4;
      CALL CRT$WRITE(CHAR(I));
      END;
   IF CHAR(5) = 'M'
      THEN CALL CRT$PRINT$STRING(.MLS);
      ELSE CALL CRT$PRINT$STRING(.YDS);
   END DISPLAY$RANGE;
```

```
/*****************************************************************************
*
* DISPLAY$TIME:
*   THIS PROCEDURE IS USED TO DISPLAY THE VALUE OF THE TIME.
*
* PARAMETERS:
*   - A. - POINTER TO A 3 BYTE VECTOR CONTAINING THE TIME VALUE.
*
*****************************************************************************/
DISPLAY$TIME: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       DIGIT BASED A BYTE,
       CHAR BYTE;
   DCL TIME (12) BYTE DATA ('TIME:      $$');

   CALL CRT$PRINT$STRING(.TIME);
   CHAR = DIGIT/10 + 30H;
   CALL CRT$WRITE(CHAR);
   CHAR = DIGIT MOD 10 + 30H;
   CALL CRT$WRITE(CHAR);
   A = A + 1;
   CALL CRT$WRITE(COLON);
   CHAR = DIGIT/10 + 30H;
   CALL CRT$WRITE(CHAR);
   CHAR = DIGIT MOD 10 + 30H;
   CALL CRT$WRITE(CHAR);
   A = A + 1;
   CALL CRT$WRITE(COLON);
   CHAR = DIGIT/10 + 30H;
   CALL CRT$WRITE(CHAR);
   CHAR = DIGIT MOD 10 + 30H;
```

```
    CALL CRT$WRITE(CHAR);
    END DISPLAY$TIME;
```

```
/*****************************************************************************
*
* DISPLAY$ORIGIN:
*   THIS PROCEDURE IS USED TO DISPLAY INFORMATION ABOUT THE COORDINATE GRID
*   ORIGIN.
*
*****************************************************************************/
DISPLAY$ORIGIN: PROCEDURE PUBLIC;
   DCL MSG (*) BYTE DATA
           ('THE COORDINATE GRID ORIGIN VALUES ARE:$$');

   CALL CRT$PRINT$STRING(.TITLE$0);
   CALL SEND$CRLF;
   CALL CRT$PRINT$STRING(.MSG);
   CALL SEND$CRLF;
   CALL DISPLAY$LAT$LONG(0, .SYSTEM.LAT);
   CALL SEND$CRLF;
   CALL DISPLAY$LAT$LONG(1, .SYSTEM.LONG);
   CALL SEND$CRLF;
   CALL SEND$CRLF;
   CALL CHECK$GO$KEY;
   CALL CLEAR$LOW$SCREEN;
   END DISPLAY$ORIGIN;
```

```
/*****************************************************************************
*
* DISPLAY$SCALE:
*   THIS PROCEDURE IS USED TO DISPLAY INFORMATION ABOUT THE SCALE BEING USED
*   IN THE GRAPHICS DISPLAY.
*
*****************************************************************************/
DISPLAY$SCALE: PROCEDURE PUBLIC;
   DCL CHAR (6) BYTE,
       (I, TEMP) BYTE;
   DCL MSG (*) BYTE DATA
            ('THE VALUE OF THE GRAPHICS SCALE IS:  $$');

   CALL CRT$PRINT$STRING(.TITLE$1);
   CALL SEND$CRLF;
   CALL SEND$CRLF;
   CALL CRT$PRINT$STRING(.MSG);
   TEMP = FP$FORMAT(.SYSTEM.SCALE, .CHAR, 2, 2);
   DO I = 0 TO 3;
      IF I = 2 THEN CALL CRT$WRITE(POINT);
      CALL CRT$WRITE(CHAR(I));
      END;
   CALL SEND$CRLF;
   CALL SEND$CRLF;
   CALL CHECK$GO$KEY;
   CALL CLEAR$LOW$SCREEN;
   END DISPLAY$SCALE;
```

```
/*****************************************************************************
*
* DISPLAY$OWN$SHIP:
*   THIS PROCEDURE IS USED TO DISPLAY INFORMATION ABOUT THE OWN SHIP.
*
*****************************************************************************/
DISPLAY$OWN$SHIP: PROCEDURE PUBLIC;
   DCL POINTER BYTE;
   CALL CRT$PRINT$STRING(.TITLE$2);
   CALL SEND$CRLF;
   CALL CRT$PRINT$STRING(.MSG$00);
   CALL SEND$CRLF;
   CALL DISPLAY$LAT$LONG(0, .OWN$SHIP$INFO.LAT);
   CALL SEND$SPACE(12);
   CALL DISPLAY$LAT$LONG(1, .OWN$SHIP$INFO.LONG);
   CALL SEND$CRLF;
   POINTER = OWN$SHIP$INFO.POINTER;
   CALL DISPLAY$XY(0, .OWN$SHIP(POINTER).X);
   CALL SEND$SPACE(18);
   CALL DISPLAY$XY(1, .OWN$SHIP(POINTER).Y);
   CALL SEND$CRLF;
   CALL SEND$CRLF;
   CALL CHECK$GO$KEY;
   CALL CLEAR$LOW$SCREEN;
   CALL CRT$PRINT$STRING(.TITLE$2);
   CALL SEND$CRLF;
   CALL CRT$PRINT$STRING(.MSG$01);
   CALL DISPLAY$TIME(.OWN$SHIP(POINTER).TIME);
   CALL SEND$CRLF;
   CALL DISPLAY$CRS$BRG(0, .OWN$SHIP(POINTER).CRS);
   CALL SEND$SPACE(20);
```

```
    CALL DISPLAY$SPD(.OWN$SHIP(POINTER).SPD);
    CALL SEND$CRLF;
    CALL SEND$CRLF;
    CALL CHECK$GO$KEY;
    CALL CLEAR$LOW$SCREEN;
    END DISPLAY$OWN$SHIP;
```

```
/*****************************************************************************
*
* DISPLAY$CONTACT$INFO:
*   THIS PROCEDURE IS USED TO DISPLAY INFORMATION ABOUT ANY CONTACT BEING
*   HANDLED BY THE SYSTEM.
*
*****************************************************************************/
DISPLAY$CONTACT$INFO: PROCEDURE PUBLIC;
   DCL DESIG ADDRESS,
       LAT (4) BYTE,
       LONG (4) BYTE,
       TIME$1 (4) BYTE , TIME$2 (4) BYTE ,
       X$DELTA (4) BYTE, Y$DELTA (4) BYTE,
       COS$DIST (4) BYTE, SIN$DIST (4) BYTE,
       BRG (4) BYTE, RNG (4) BYTE,
       T (3) BYTE, DISTANCE (4) BYTE,
       STRING (23) BYTE,
       (TEMP, OK, INDEX, I, POINTER) BYTE;
   DCL DEG$TO$RAD (4) BYTE DATA (035H, 0FAH, 08EH, 03CH); /* 0.0174532925 */

   DCL MSG0 (*) BYTE DATA
            ('NO COURSE INFORMATION AVAILABLE.$$'),
       MSG1 (*) BYTE DATA
            ('NO SPEED INFORMATION AVAILABLE.$$'),
       MSG2 (*) BYTE DATA
            ('NO CPA INFORMATION AVAILABLE.$$'),
       MSG3 (*) BYTE DATA
            ('ENTER CONTACT DESIG AS REQUESTED:$$'),
       MSG4 (*) BYTE DATA
            ('DESIG NOT IN USE.$$'),
       MSG5 (*) BYTE DATA
```

```
          ('CURRENT NUMBER OF POSITIONS:  $$'),
     MSG6 (*) BYTE DATA
          ('THE ACTUAL ESTIMATED POSITION IS:$$');

T(0) = HOURS;   /* SAVE TIME OF CALL */
T(1) = MINUTES;
T(2) = SECONDS;
OK = 0;
DO WHILE OK = 0;
   CALL CRT$PRINT$STRING(.TITLE$3);
   CALL SEND$CRLF;
   CALL CRT$PRINT$STRING(.MSG3);
   CALL SEND$CRLF;
   DESIG = GET$DESIG;
   INDEX = CHECK$DESIG(DESIG);
   IF INDEX = 0FFH
   THEN DO;
        CALL CRT$PRINT$STRING(.MSG4);
        CALL SEND$CRLF;
        CALL CHECK$GO$KEY;
        END;
   ELSE DO;
        OK = 1;
        END;
   CALL CLEAR$LOW$SCREEN;
   END;
CALL CRT$PRINT$STRING(.TITLE$3);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(.MSG$03);
CALL SEND$CRLF;
CALL DISPLAY$DESIG(DESIG);
```

```
   CALL SEND$SPACE(15);
   CALL DISPLAY$TYPE(.CONTACT$INFO(INDEX).TYPE);
   CALL SEND$SPACE(6);
   CALL DISPLAY$CLASS(.CONTACT$INFO(INDEX).KIND);
   CALL SEND$CRLF;
   CALL CRT$PRINT$STRING(.MSG5);
   IF CONTACT$INFO(INDEX).FLAG
      THEN TEMP = 15;
      ELSE TEMP = (CONTACT$INFO(INDEX).POINTER  MOD 15) + 1;
   CALL BYTE$CHAR(TEMP);
   CALL SEND$CRLF;
   CALL SEND$CRLF;
   CALL CHECK$GO$KEY;
   CALL CLEAR$LOW$SCREEN;
   CALL CRT$PRINT$STRING(.TITLE$3);
   CALL SEND$CRLF;
   CALL CRT$PRINT$STRING(.MSG$00);
   CALL SEND$CRLF;
   POINTER = CONTACT$INFO(INDEX).POINTER;
   CALL CONV$LAT$LONG(.CONTACT$POSI(POINTER).X, .CONTACT$POSI(POINTER).Y,
                      .LAT, .LONG);
   CALL DISPLAY$LAT$LONG(0, .LAT);
   CALL SEND$SPACE(12);
   CALL DISPLAY$LAT$LONG(1, .LONG);
   CALL SEND$CRLF;
   CALL DISPLAY$XY(0, .CONTACT$POSI(POINTER).X);
   CALL SEND$SPACE(18);
   CALL DISPLAY$XY(1, .CONTACT$POSI(POINTER).Y);
   CALL SEND$CRLF;
   CALL SEND$CRLF;
   CALL CHECK$GO$KEY;
```

```
   CALL CLEAR$LOW$SCREEN;
   TEMP = GET$CPA(INDEX, .STRING);
   CALL CRT$PRINT$STRING(.TITLE$3);
   CALL SEND$CRLF;
   CALL CRT$PRINT$STRING(.MSG$01);
   CALL DISPLAY$TIME(.CONTACT$POSI(POINTER).TIME);
   CALL SEND$CRLF;
   CALL DISPLAY$CRS$BRG(1, .CONTACT$POSI(POINTER).BRG);
   CALL SEND$SPACE(20);
   CALL DISPLAY$RANGE(.CONTACT$POSI(POINTER).RNG);
   CALL SEND$CRLF;
   IF CONTACT$INFO(INDEX).CRS$FLAG
   THEN DO;
        CALL DISPLAY$CRS$BRG(0, .CONTACT$POSI(POINTER).CRS);
        CALL SEND$SPACE(20);
        END;
   ELSE DO;
        CALL CRT$PRINT$STRING(.MSG0);
        CALL SEND$SPACE(3);
        END;
   IF CONTACT$INFO(INDEX).SPD$FLAG
      THEN CALL DISPLAY$SPD(.CONTACT$POSI(POINTER).SPD);
      ELSE CALL CRT$PRINT$STRING(.MSG1);
   CALL SEND$CRLF;
   CALL SEND$CRLF;
   CALL CHECK$GO$KEY;
   CALL CLEAR$LOW$SCREEN;
   CALL CRT$PRINT$STRING(.TITLE$3);
   CALL SEND$CRLF;
   CALL CRT$PRINT$STRING(.MSG$02);
   CALL SEND$CRLF;
```

```
IF TEMP = 0
THEN DO;
     CALL CRT$PRINT$STRING(.MSG2);
     CALL SEND$CRLF;
     END;
ELSE DO;
     IF STRING(14) < 3AH
     THEN DO;
          CALL CRT$PRINT$STRING(.('TIME:      $$'));
          DO I = 7 TO 10;
             IF I = 9 THEN CALL CRT$WRITE(COLON);
             CALL CRT$WRITE(STRING(I));
             END;
          CALL SEND$SPACE(9);
          CALL CRT$PRINT$STRING(.('BEARING:   $$'));
          DO I = 11 TO 14;
             IF I = 14 THEN CALL CRT$WRITE(POINT);
             CALL CRT$WRITE(STRING(I));
             END;
          CALL SEND$SPACE(9);
          CALL CRT$PRINT$STRING(.('RANGE:     $$'));
          DO I = 15 TO 19;
             CALL CRT$WRITE(STRING(I));
             END;
          IF STRING(20) = 'M'
             THEN CALL CRT$PRINT$STRING(.(' MILES$$'));
             ELSE CALL CRT$PRINT$STRING(.(' YARDS$$'));
          CALL SEND$CRLF;
          END;
     ELSE DO;
          CALL SEND$SPACE(30);
```

```
            IF STRING(14) = 'L'
            THEN DO;
                 CALL SEND$SPACE(5);
                 CALL START$BLINK;
                 CALL CRT$PRINT$STRING(.('COLLISION AT $$'));
                 STRING(12), STRING(13) = '$';
                 STRING(11) = STRING(10);
                 STRING(10) = STRING(9);
                 STRING(9) = COLON;
                 CALL CRT$PRINT$STRING(.STRING(7));
                 CALL CRT$WRITE(18H);          /* STOP BLINK */
                 END;
            ELSE DO;
                 IF STRING(14) = 'A'
                    THEN CALL CRT$PRINT$STRING(.('     MOVING AWAY.$$'));
                    ELSE CALL CRT$PRINT$STRING(.('SAME COURSE AND SPEED.$$'));
                 END;
            END;
       END;
  CALL SEND$CRLF;
  CALL CHECK$GO$KEY;
  CALL CLEAR$LOW$SCREEN;
  CALL CRT$PRINT$STRING(.TITLE$3);
  CALL SEND$CRLF;
  CALL CRT$PRINT$STRING(.MSG6);
  CALL SEND$CRLF;
  IF CONTACT$INFO(INDEX).CRS$FLAG AND
     CONTACT$INFO(INDEX).SPD$FLAG
  THEN DO;
       IF T(0) < CONTACT$POSI(POINTER).TIME(0)
          THEN T(0) = T(0) + 24;
```

```
      /* FIND ESTIMATED TIME */
   CALL CONV$CONTACT$TIME(.T, .TIME$1);
   CALL CONV$CONTACT$TIME(.CONTACT$POSI(POINTER).TIME, .TIME$2);
   CALL FSUB(.TIME$1, .TIME$2, .TIME$1);
      /* FIND ESTIMATED DISTANCE TRAVELED BY CONTACT */
   CALL FMUL(.CONTACT$POSI(POINTER).SPD, .TIME$1, .DISTANCE);
   CALL FMUL(.DEG$TO$RAD, .CONTACT$POSI(POINTER).CRS, .BRG);
   CALL COS$SIN(.BRG, .COS$DIST, .SIN$DIST);
   CALL FMUL(.DISTANCE, .COS$DIST, .Y$DELTA);
   CALL FMUL(.DISTANCE, .SIN$DIST, .X$DELTA);
   CALL FADD(.Y$DELTA, .CONTACT$POSI(POINTER).Y, .Y$DELTA);
   CALL FADD(.X$DELTA, .CONTACT$POSI(POINTER).X, .X$DELTA);
   TEMP = OWN$SHIP$INFO.POINTER;
   CALL FSUB(.Y$DELTA, .OWN$SHIP(TEMP).Y, .Y$DELTA);
   CALL FSUB(.X$DELTA, .OWN$SHIP(TEMP).X, .X$DELTA);
      /* FIND ESTIMATED BEARING */
   CALL ARC$TAN(.Y$DELTA, .X$DELTA, .BRG);
   CALL FDIV(.BRG, .DEG$TO$RAD, .BRG);
      /* FIND ESTIMATED RANGE */
   CALL FSQR(.Y$DELTA, .Y$DELTA);
   CALL FSQR(.X$DELTA, .X$DELTA);
   CALL FADD(.Y$DELTA, .X$DELTA, .RNG);
   CALL FSQRT(.RNG, .RNG);
      /* PRINT ESTIMATED VALUES */
   CALL DISPLAY$CRS$BRG(1, .BRG);
   CALL SEND$SPACE(20);
   CALL DISPLAY$RANGE(.RNG);
   END;
ELSE DO;
   CALL CRT$PRINT$STRING(.MSG0);
   CALL SEND$SPACE(3);
```

```
        CALL CRT$PRINT$STRING(.MSG1);
        END;
    CALL SEND$CRLF;
    CALL SEND$CRLF;
    CALL CHECK$GO$KEY;
    CALL CLEAR$LOW$SCREEN;
    END DISPLAY$CONTACT$INFO;
```

```
/*****************************************************************************
*
* DISPLAY$SYSTEM:
*   THIS PROCEDURE IS USED TO DISPLAY THE DESIGNATIONS OF ALL THE CONTACTS
*   THAT ARE BEING MAINTAINED BY THE SYSTEM.
*
*****************************************************************************/
DISPLAY$SYSTEM: PROCEDURE PUBLIC;
   DCL BUFFER (18) BYTE,
       I BYTE;
   DCL M0 (*) BYTE DATA
         ('THE FOLLOWING CONTACTS ARE BEING MAINTAINED BY THE SYSTEM:$$');

   DO I = 0 TO LAST(BUFFER);
      BUFFER(I) = ' ';
      END;
   BUFFER(16), BUFFER(17) = '$';
   CALL CRT$PRINT$STRING(.TITLE$4);
   CALL SEND$CRLF;
   CALL CRT$PRINT$STRING(.M0);
   CALL SEND$CRLF;
   DO I = 0 TO 14;
      IF CONTACT$INFO(I).DESIG <> 00H
      THEN DO;
           CALL DE$HASH(CONTACT$INFO(I).DESIG, .BUFFER(10));
           CALL CRT$PRINT$STRING(.BUFFER);
           END;
      END;
   CALL SEND$CRLF;
   CALL CHECK$GO$KEY;
   CALL CLEAR$LOW$SCREEN;
```

```
END DISPLAY$SYSTEM;
```

```
/*****************************************************************************
*
* DISPLAY$SAFE$RNG:
*    THIS PROCEDURE IS USED TO PRESENT TO THE OPERATOR THE CURRENT VALUE OF
*    THE SAFE C.P.A.  RANGE.
*
*****************************************************************************/
DISPLAY$SAFE$RNG: PROCEDURE PUBLIC;
    CALL CRT$PRINT$STRING(.TITLE$5);
    CALL SEND$CRLF;
    CALL SEND$CRLF;
    CALL DISPLAY$RANGE(.SAFE$RNG);
    CALL SEND$CRLF;
    CALL SEND$CRLF;
    CALL CHECK$GO$KEY;
    CALL CLEAR$LOW$SCREEN;
    END DISPLAY$SAFE$RNG;
```

```
/*****************************************************************************
*
* DISPLAY$WIND:
*   THIS PROCEDURE IS USED TO DISPLAY INFORMATION ABOUT THE WIND.
*
*****************************************************************************/
DISPLAY$WIND: PROCEDURE PUBLIC;
   DCL STRING (6) BYTE,
       (TEMP, I) BYTE;
   DCL MSG0 (*) BYTE DATA ('WIND DIRECTION:  $$'),
       MSG1 (*) BYTE DATA ('WIND SPEED:       $$');

   CALL CRT$PRINT$STRING(.TITLE$6);
   CALL SEND$CRLF;
   CALL SEND$CRLF;
   CALL CRT$PRINT$STRING(.MSG0);
   TEMP = FP$FORMAT(.SYSTEM.WIND$DIR, .STRING, 3, 1);
   DO I = 0 TO 3;
      IF I = 3 THEN CALL CRT$WRITE(POINT);
      CALL CRT$WRITE(STRING(I));
      END;
   CALL SEND$CRLF;
   CALL CRT$PRINT$STRING(.MSG1);
   TEMP = FP$FORMAT(.SYSTEM.WIND$SPD, .STRING, 2, 1);
   DO I = 0 TO 2;
      IF I = 2 THEN CALL CRT$WRITE(POINT);
      CALL CRT$WRITE(STRING(I));
      END;
   CALL SEND$CRLF;
   CALL SEND$CRLF;
   CALL CHECK$GO$KEY;
```

```
    CALL CLEAR$LOW$SCREEN;
    END DISPLAY$WIND;
```

```
/*****************************************************************************
*
* DISPLAY$UPDATE$TIME:
*   THIS PROCEDURE IS USED TO DISPLAY THE TIME BETWEEN UPDATES.
*
*****************************************************************************/
DISPLAY$UPDATE$TIME: PROCEDURE (TEMP$TIME) PUBLIC;
   DCL TEMP$TIME BYTE;
   DCL M0 (*) BYTE DATA
          ('THE TIME BETWEEN UPDATES IS $$'),
       M1 (*) BYTE DATA
          (' SECONDS.$$');

   CALL CRT$PRINT$STRING(.TITLE$7);
   CALL SEND$CRLF;
   CALL CRT$PRINT$STRING(.M0);
   CALL BYTE$CHAR(TEMP$TIME);
   CALL CRT$PRINT$STRING(.M1);
   CALL SEND$CRLF;
   CALL SEND$CRLF;
   CALL CHECK$GO$KEY;
   CALL CLEAR$LOW$SCREEN;
   END DISPLAY$UPDATE$TIME;


END DISPLAY$CMDS;
```

```
COMMANDS: DO;


CRT$READ:
   PROCEDURE BYTE EXTERNAL;
   END;

CRT$PRINT$STRING:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

CRT$WRITE:
   PROCEDURE (A) EXTERNAL;
   DECLARE A BYTE; END;

SEND$CR:
   PROCEDURE EXTERNAL;
   END;

SEND$BEL:
   PROCEDURE EXTERNAL;
   END;

SEND$BS:
   PROCEDURE EXTERNAL;
   END;

SEND$SUB:
   PROCEDURE EXTERNAL;
```

```
    END;

SEND$CRLF:
    PROCEDURE EXTERNAL;
    END;

GET$STRING:
    PROCEDURE (A,B) EXTERNAL;
    DECLARE A ADDRESS,B BYTE; END;

PUT$NUMBER$BUFFER:
    PROCEDURE (A,B) EXTERNAL;
    DECLARE A BYTE, B ADDRESS; END;

ASCII$TO$FLOAT:
    PROCEDURE (A,B,C) EXTERNAL;
    DECLARE (A,C) ADDRESS, B BYTE; END;

FLOAT$TO$ASCII:
    PROCEDURE (A,B,C) EXTERNAL;
    DECLARE (A,B,C) ADDRESS; END;

CLEAR$LOW$SCREEN:
    PROCEDURE EXTERNAL;
    END;

FADD:
    PROCEDURE (A,B,C) EXTERNAL;
    DECLARE (A,B,C) ADDRESS; END;

FSUB:
```

COMMANDS

```
    PROCEDURE (A,B,C) EXTERNAL;
    DECLARE (A,B,C) ADDRESS; END;

FMUL:
    PROCEDURE (A,B,C) EXTERNAL;
    DECLARE (A,B,C) ADDRESS; END;

FDIV:
    PROCEDURE (A,B,C) EXTERNAL;
    DECLARE (A,B,C) ADDRESS; END;

EDIV:
    PROCEDURE (A,B,C,D) EXTERNAL;
    DECLARE (A,B,C,D) ADDRESS; END;

FLTDS:
    PROCEDURE (A,B) EXTERNAL;
    DECLARE (A,B) ADDRESS; END;

FIXSD:
    PROCEDURE (A,B) EXTERNAL;
    DECLARE (A,B) ADDRESS; END;

FCMPR:
    PROCEDURE (A,B,C) BYTE EXTERNAL;
    DECLARE (A,B) ADDRESS, C BYTE; END;

FZTST:
    PROCEDURE (A,B) BYTE EXTERNAL;
    DECLARE (A,B) ADDRESS; END;
```

```
DECLARE LIT LITERALLY 'LITERALLY',
        DCL LIT 'DECLARE';

DCL POINT LIT '2EH',                    /* DECIMAL POINT.  */
    ETEOL LIT '17H';                    /* ERASE TO END OF LINE.  */

DCL
    FP$0$25 (4) BYTE DATA (00H,00H,80H,3EH),
    FP$2 (4) BYTE DATA (00H,00H,00H,40H),
    FP$5 (4) BYTE DATA (00H,00H,0A0H,040H),
    FP$25 (4) BYTE DATA (00H,00H,0C8H,41H),
    FP$60 (4) BYTE DATA (00H,00H,70H,42H),
    FP$90 (4) BYTE DATA (00H,00H,0B4H,42H),
    FP$100 (4) BYTE DATA (00H,00H,0C8H,042H),
    FP$180 (4) BYTE DATA (00H,00H,34H,43H),
    FP$360 (4) BYTE DATA (0DFH,0FFH,0B3H,043H),
    FP$2000 (4) BYTE DATA (0E4H,2BH,0FDH,044H),          /* 2025.3716 */
    FP$202500 (4) BYTE DATA (04AH,0CAH,045H,048H),       /* 202537.15625 */
    FP$59$9 (4) BYTE DATA (9AH,99H,6FH,42H),
    FP$99$9 (4) BYTE DATA (0CDH,0CCH,0C7H,042H),
    FP$MIN$TO$RAD (4) BYTE DATA (98H,82H,98H,39H);      /* 0.00029089 */
```

```
/*****************************************************************************
*
* PRINT$ERROR$MSG:
*   THIS PROCEDURE IS USED TO PRINT AN ERROR MESSAGE FOR INVALID INPUT.
*   IT WILL ALSO RETURN THE CURSOR TO THE BEGINNING OF THE SAME LINE.
*
*****************************************************************************/
PRINT$ERROR$MSG: PROCEDURE;
   DCL MSG(*) BYTE DATA (' ***  BAD FORMAT.   ***$$');
   CALL SEND$BEL;
   CALL CRT$PRINT$STRING(.MSG);
   CALL SEND$CR;
   CALL SEND$SUB;
   END PRINT$ERROR$MSG;
```

```
/*****************************************************************************
*
* CHECK$YES$NO:
*   PROCEDURE USED TO CHECK FOR A VALID YES/NO INPUT FROM THE CRT.
*
* USAGE:
*   TYPED PROCEDURE.  A VALUE OF 1 WILL BE RETURNED IF THE ANSWER IS 'YES',
*   OTHERWISE, THE VALUE RETURNED IS 0.
*
*****************************************************************************/
CHECK$YES$NO: PROCEDURE BYTE PUBLIC;
   DCL CHAR BYTE;
   CHAR = CRT$READ;
   DO WHILE (CHAR <> 'Y') AND (CHAR <> 'N')      /* UPPER CASE.  */
         AND (CHAR <> 'y') AND (CHAR <> 'n');    /* LOWER CASE.  */
      CALL SEND$BEL;
      CHAR = CRT$READ;
      END;
   IF (CHAR = 'Y') OR (CHAR = 'y')
   THEN DO;
         CALL CRT$PRINT$STRING(.('YES$$'));
         RETURN 1;
         END;
   ELSE DO;
         CALL CRT$PRINT$STRING(.('NO $$'));
         RETURN 0;
         END;
    END CHECK$YES$NO;
```

```
/*****************************************************************************
*
* CHECK$FP$VALUE:
*   THIS PROCEDURE IS USED TO CHECK A GIVEN VALUE AGAINST A GIVEN LIMIT.
*
* PARAMETERS:
*   - A. - POINTER TO A FP VALUE.
*   - B. - POINTER TO A FP VALUE DENOTING THE LIMIT.
*
* USAGE:
*   TYPED PROCEDURE.  A VALUE OF 00H WILL BE RETURNED IF THE VALUE IS GREATER
*   THAN THE GIVEN LIMIT.  OTHERWISE A VALUE OF 001H WILL BE RETURNED.
*
*****************************************************************************/
CHECK$FP$VALUE: PROCEDURE (A,B) BYTE PUBLIC;
   DCL (A,B) ADDRESS,
       (CHECK, TWO) BYTE;
   TWO = 2;
   IF (CHECK:= FCMPR(A,B,.TWO))
   THEN DO;
        CALL PRINT$ERROR$MSG;
        RETURN 0;
        END;
   ELSE DO;
        CALL CRT$WRITE(ETEOL);              /* ERASE TO THE END OF LINE.  */
        RETURN 1;
        END;
   END CHECK$FP$VALUE;
```

```
/*****************************************************************************
*
* CHECK$INPUT:
*    PROCEDURE USED TO CHECK THE VALIDITY OF THE INPUT PRESENT AT THAT MOMENT
*    IN THE SCREEN.
*
*****************************************************************************/
CHECK$INPUT: PROCEDURE BYTE PUBLIC;
    DCL CHAR BYTE;
    CALL CRT$PRINT$STRING(.('IS THE INPUT CORRECT? (Y/N)  $$'));
    CHAR = CHECK$YES$NO;
    RETURN CHAR;
    END CHECK$INPUT;
```

```
/*****************************************************************************
*
* GET$DEGREES:
*   THIS PROCEDURE IS USED TO OBTAIN 2 OR 3 NUMERIC CHARACTERS FROM THE CRT
*   THAT REPRESENT A VALUE IN DEGREES OF LATITUDE OR LONGITUDE RESPECTIVELY.
*
* PARAMETERS:
*   - NUM. - NUMBER OF NUMERIC CHARACTERS DESIRED.  (2 OR 3 ONLY)
*   - A. - POINTER TO A MEMORY LOCATION IN WHICH THE RESULT IS DESIRED.
*
* USAGE:
*   THE FLOATING POINT REPRESENTATION OF THE DEGREES, WILL BE RETURNED CON-
*   VERTED TO MINUTES OF LAT OR LONG.
*
*****************************************************************************/
GET$DEGREES: PROCEDURE (NUM,A) PUBLIC;
   DCL A ADDRESS,
       NUMBER BASED A (4) BYTE,
       (NUM, OK) BYTE,
       BUFFER(6) BYTE;
   BUFFER(0), BUFFER(1) = NUM;
   BUFFER(NUM + 2) = 0;
   OK = 0;
   DO WHILE OK = 0;
      CALL CRT$PRINT$STRING(.('DEGREES:  $$'));
      CALL PUT$NUMBER$BUFFER(NUM,.BUFFER(2));
      CALL ASCII$TO$FLOAT(.BUFFER,NUM + 3,.NUMBER);
      IF NUM = 2
         THEN OK = CHECK$FP$VALUE(.NUMBER,.FP$90);
         ELSE OK = CHECK$FP$VALUE(.NUMBER,.FP$180);
      END;
```

```
CALL FMUL(.NUMBER,.FP$60,.NUMBER);
END GET$DEGREES;
```

```
/*****************************************************************************
*
* GET$MINUTES:
*   THIS PROCEDURE IS USED TO GET FROM THE CRT, THREE NUMERIC CHARACTERS RE-
*   PRESENTING THE VALUE OF MINUTES.  THIS PROCEDURE WILL PROMPT FOR TWO
*   INTEGERS AND ONE DECIMAL VALUE.
*
* PARAMETERS:
*   - A. - POINTER TO A MEMORY LOCATION IN WHICH THE FP REPRESENTATION OF THE
*     CHARACTERS OBTAINED, IS DESIRED TO BE PLACED.
*
*****************************************************************************/
GET$MINUTES: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       NUMBER BASED A (4) BYTE,
       BUFFER(6) BYTE,
       OK BYTE;
   BUFFER(0) = 3;
   BUFFER(1) = 2;
   BUFFER(5) = 0;
   OK = 0;
   DO WHILE OK = 0;
      CALL CRT$PRINT$STRING(.('MINUTES:  $$'));
      CALL PUT$NUMBER$BUFFER(2,.BUFFER(2));
      CALL CRT$WRITE(POINT);
      CALL PUT$NUMBER$BUFFER(1,.BUFFER(4));
      CALL ASCII$TO$FLOAT(.BUFFER,6,.NUMBER);
      OK = CHECK$FP$VALUE(.NUMBER,.FP$59$9);
      END;
   END GET$MINUTES;
```

```
/*****************************************************************************
*
* GET$SIGN:
*   THIS PROCEDURE IS USED TO GET FROM THE CRT, A CHARACTER THAT WILL REPRE-
*   SENT THE N/S LATITUDE OR E/W LONGITUDE.
*
* PARAMETERS:
*   - T1,T2. - ASCII CHARACTERS REPRESENTING THE VALUES AGAINST WHICH THE
*     INPUT FROM THE CRT IS DESIRED TO BE COMPARED.
*
* USAGE:
*   TYPED PROCEDURE.  A VALUE OF 1 WILL BE RETURNED IF THE INPUT FROM THE CRT
*   IS EQUAL TO THE VALUE OF T1, OTHERWISE A VALUE OF 0 WILL BE RETURNED.
*
*****************************************************************************/
GET$SIGN: PROCEDURE (T1,T2) BYTE PUBLIC;
   DCL (T1,T2,SIGN) BYTE;
   SIGN = 0;
   DO WHILE (SIGN <> T1) AND (SIGN <> T2);
      CALL CRT$PRINT$STRING(.('SIGN:  $$'));
      CALL GET$STRING(.SIGN,1);
      IF (SIGN <> T1) AND (SIGN <> T2)
      THEN CALL PRINT$ERROR$MSG;
      ELSE CALL CRT$WRITE(ETEOL);
      END;
   IF SIGN = T1 THEN RETURN 1;
                ELSE RETURN 0;
   END GET$SIGN;
```

```
/*****************************************************************************
*
* FP$FORMAT:
*   THIS PROCEDURE IS USED TO OBTAIN AN SPECIFIED NUMBER OF ASCII CHARAC-
*   TERS REPRESENTING A NUMERIC VALUE IN FP FORMAT.
*
* PARAMETERS:
*   - A. - POINTER TO A 4 BYTE VECTOR CONTAINING A FP NUMBER.
*   - B. - POINTER TO A MEMORY LOCATION IN WHICH THE STRING OF ASCII CHARAC-
*     TERS IS DESIRED TO BE PLACED.
*   - NUMINT. - NUMBER OF CHARACTERS REPRESENTING THE DESIRED INTEGER
*     PORTION.
*   - NUMDEC. - SIMILAR TO NUMINT, BUT REPRESENTING THE DECIMAL PORTION
*     DESIRED.
*
* USAGE:
*   TYPED PROCEDURE.  IF THE SIGN OF THE GIVEN FP NUMBER IS POSITIVE, A
*   VALUE OF 0 IS RETURNED, OTHERWISE, A 1 IS RETURNED.
*
*****************************************************************************/
FP$FORMAT: PROCEDURE (A,B,NUMINT,NUMDEC) BYTE PUBLIC;
   DCL (A,B) ADDRESS,
       FP$NUM BASED A (4) BYTE,
       STRING BASED B (16) BYTE,
       BUFFER (28) BYTE,
       (NUMINT,NUMDEC,NUM,SIGN,I,J,N,N1) BYTE;
   N1 = 0;
   DO I = 0 TO NUMINT + NUMDEC;
      STRING (I) = '0';
      END;
   CALL FLOAT$TO$ASCII(.FP$NUM,.BUFFER,.NUM);
```

```
DO I = NUM TO 27;
   BUFFER(I) = '0';
   END;
IF BUFFER(0) = ' '
   THEN SIGN = 0;
   ELSE SIGN = 1;
I = 1;
DO WHILE BUFFER(I) <> POINT;
   N1 = N1 + 1;
   I = I + 1;
   END;
IF N1 > NUMINT
THEN DO;
     CALL CRT$PRINT$STRING(.('NUMBER TOO LARGE.  $$'));
     HALT;
     END;
I = 1;
J = NUMINT - N1;
N = NUMINT + NUMDEC;
DO WHILE N > 0;
   IF BUFFER(I) = POINT THEN I = I + 1;
   STRING(J) = BUFFER(I);
   J = J + 1;
   I = I + 1;
   N = N - 1;
   END;
IF (BUFFER(I) >= '5') AND (STRING(J-1) <> '9')      /* TO "ROUND". */
   THEN STRING(J-1) = STRING(J-1) + 1;
N = NUMINT + NUMDEC;
STRING(N), STRING (N + 1) = '$';
RETURN SIGN;
```

```
END FP$FORMAT;
```

```
/*****************************************************************************
*
* RANGE$FORMAT:
*   THIS PROCEDURE IS USED TO OBTAIN IN ASCII CHARACTERS, THE VALUE OF A
*   FP NUMBER REPRESENTING A RANGE.  IF THE VALUE IS LESS THAN OR EQUAL TO 5
*   MILES, THEN THE RANGE WILL BE GIVEN IN YARDS, OTHERWISE, IT WILL BE GIVEN
*   IN MILES.
*
* PARAMETERS:
*   - A. - POINTER TO A 4 BYTE VECTOR REPRESENTING A FP NUMBER.
*   - B. - POINTER TO A MEMORY LOCATION IN WHICH THE STRING IS DESIRED TO BE
*     PLACED.
*
*****************************************************************************/
RANGE$FORMAT: PROCEDURE (A,B) PUBLIC;
   DCL (A,B) ADDRESS,
       VALUE BASED A (4) BYTE,
       STRING BASED B (6) BYTE,
       RESULT (4) BYTE,
       (TEMP, TWO) BYTE;
   TWO = 2;
   IF (TEMP:= FCMPR(A,.FP$5,.TWO))
   THEN DO;
        TEMP = FP$FORMAT(.VALUE,.STRING(0),3,1);
        STRING(4) = STRING(3);
        STRING(3) = POINT;
        STRING(5) = 'M';
        END;
   ELSE DO;
        CALL FMUL(.VALUE,.FP$2000,.RESULT);
        TEMP = FP$FORMAT(.RESULT,.STRING(0),5,0);
```

```
        STRING(5) = 'Y';
        END;
    END RANGE$FORMAT;
```

```
/*****************************************************************************
*
* LAT$LONG$FORMAT:
*   THIS PROCEDURE IS USED TO CONVERT A FLOATING POINT VALUE REPRESENTATION
*   OF LAT/LONG IN MINUTES, INTO A CORRESPONDING STRING OF CHARACTERS.
*
* PARAMETERS:
*   - A. - POINTER TO A MEMORY LOCATION CONTAINING A FP VALUE.
*   - B. - POINTER TO A MEMORY LOCATION IN WHICH THE STRING OF ASCII CHARAC-
*     TERS IS DESIRED TO BE PLACED.
*   - TYPE. - CAN HAVE ONE OF TWO VALUES:  0 ---> LATITUDE DESIRED.
*                                          1 ---> LONGITUDE DESIRED.
*
*****************************************************************************/
LAT$LONG$FORMAT: PROCEDURE (A,B,TYPE) PUBLIC;
   DCL (A,B) ADDRESS,
       SIXTY ADDRESS DATA (060),
       LAT$LONG BASED A (4) BYTE,
       STRING BASED B (9) BYTE,
       BUFFER(28) BYTE,
       VALUE (4) BYTE,
       TEMP (4) BYTE,
       TEMP1 (4) BYTE,
       REM (4) BYTE,
       (NUM,I,SIGN,TYPE,J,TEST) BYTE;
   SIGN = 0;
   DO I = 0 TO 3;
      VALUE(I) = LAT$LONG(I);
      END;
   IF LAT$LONG(3) >= 080H
   THEN DO;
```

```
        SIGN = 1;
        VALUE(3) = VALUE(3) XOR 080H;
        END;
    CALL FIXSD(.VALUE,.TEMP);
    CALL EDIV(.TEMP,.SIXTY,.TEMP1,.REM);
    CALL FLTDS(.TEMP1,.TEMP1);
    TEST = FP$FORMAT(.TEMP1,.STRING,3,0);
    CALL FLTDS(.REM,.REM);
    CALL FLTDS(.TEMP,.TEMP);
    CALL FSUB(.VALUE,.TEMP,.TEMP);
    CALL FADD(.TEMP,.REM,.TEMP);
    TEST = FP$FORMAT(.TEMP,.STRING(3),2,1);
    IF TYPE = 0
    THEN DO;
        IF SIGN = 1 THEN STRING(6) = 'S';
                    ELSE STRING(6) = 'N';
        END;
    ELSE DO;
        IF SIGN = 1 THEN STRING(6) = 'W';
                    ELSE STRING(6) = 'E';
        END;
    STRING(7),STRING(8) = '$';
    END LAT$LONG$FORMAT;
```

```
/*****************************************************************************
*
* GET$TIME$ZONE:
*   THIS PROCEDURE IS USED TO OBTAIN A STRING OF ASCII CHARACTERS FROM
*   THE CRT, REPRESENTING THE VALUE AND SIGN OF THE TIME ZONE.
*
* PARAMETERS:
*   - A. - POINTER TO A MEMORY LOCATION IN WHICH THE STRING IS DESIRED TO
*     BE PLACED.
*
*****************************************************************************/
GET$TIME$ZONE: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       RESULT BASED A (5) BYTE,
       (OK, SIGN, TEMP, VALUE) BYTE;
   OK = 0;
   DO WHILE OK = 0;
      CALL CRT$PRINT$STRING(.('ENTER THE TIME ZONE VALUE AS REQUESTED:$$'));
      CALL SEND$CRLF;
      CALL CRT$PRINT$STRING(.('SIGN:  $$'));
      SIGN = CRT$READ;
      DO WHILE (SIGN <> '+') AND (SIGN <> '-');
         CALL SEND$BEL;
         SIGN = CRT$READ;
         END;
      RESULT(0) = SIGN;
      CALL CRT$WRITE(SIGN);
      CALL SEND$CRLF;
      TEMP = 0;
      DO WHILE TEMP = 0;
         CALL CRT$PRINT$STRING(.('VALUE:  $$'));
```

```
        CALL PUT$NUMBER$BUFFER(2,.RESULT(1));
        VALUE = ((RESULT(1) - 30H) * 10) + (RESULT(2) - 30H);
        IF VALUE > 12 THEN CALL PRINT$ERROR$MSG;
        ELSE DO;
             CALL CRT$WRITE(ETEOL);
             TEMP = 1;
             END;
        END;
     CALL SEND$CRLF;
     OK = CHECK$INPUT;
     CALL CLEAR$LOW$SCREEN;
     END;
  RESULT(3), RESULT(4) = '$';
  END GET$TIME$ZONE;
```

```
/*****************************************************************************
*
* GET$LAT:
*   PROCEDURE USED TO OBTAIN THE VALUES OF LATITUDE.
*
* PARAMETERS:
*   - A. - POINTER TO MEMORY LOCATION IN WHICH THE FP REPRESENTATION
*     OF THE VALUE OF THE LATITUDE IN MINUTES IS DESIRED TO BE PLACED.
*
*****************************************************************************/
GET$LAT: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       RESULT BASED A (4) BYTE,
       OP1 (4) BYTE,
       OP2 (4) BYTE,
       (SIGN,OK,TEST,FIVE) BYTE;
   FIVE = 5;
   OK = 0;
   DO WHILE OK = 0;
      CALL CRT$PRINT$STRING(.('ENTER THE LATITUDE VALUE AS REQUESTED:$$'));
      CALL SEND$CRLF;
      CALL GET$DEGREES(2,.OP1);
      CALL SEND$CRLF;
      CALL GET$MINUTES(.OP2);
      CALL SEND$CRLF;
      CALL FADD(.OP1,.OP2,.RESULT);
      SIGN = GET$SIGN('N','S');
      IF SIGN = 0
      THEN DO;
         IF (TEST:= FZTST(.RESULT,.FIVE)) THEN
            RESULT(3) = RESULT(3) XOR 080H;
```

```
            CALL CRT$PRINT$STRING(.('OUTH. $$'));
            END;
        ELSE CALL CRT$PRINT$STRING(.('ORTH. $$'));
        CALL SEND$CRLF;
        OK = CHECK$INPUT;
        CALL CLEAR$LOW$SCREEN;
        END;
    END GET$LAT;
```

```
/*****************************************************************************
*
* GET$LONG:
*   PROCEDURE USED TO OBTAIN THE FP REPRESENTATION OF THE LONGITUDE IN
*   MINUTES.
*
* PARAMETERS:
*   - A. - POINTER TO A MEMORY LOCATION IN WHICH THE VALUE OF THE
*          LONGITUDE IN MINUTES IS DESIRED TO BE STORED.
*
*****************************************************************************/
GET$LONG: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       RESULT BASED A (4) BYTE,
       OP1 (4) BYTE,
       OP2 (4) BYTE,
       (SIGN,OK,TEST,FIVE) BYTE;
   FIVE = 5;
   OK = 0;
   DO WHILE OK = 0;
      CALL CRT$PRINT$STRING(.('ENTER THE LONGITUDE VALUE AS REQUESTED:$$'));
      CALL SEND$CRLF;
      CALL GET$DEGREES(3,.OP1);
      CALL SEND$CRLF;
      CALL GET$MINUTES(.OP2);
      CALL SEND$CRLF;
      CALL FADD(.OP1,.OP2,.RESULT);
      SIGN = GET$SIGN('E','W');
      IF SIGN = 0
      THEN DO;
         CALL CRT$PRINT$STRING(.('EST$$'));
```

```
        IF (TEST:= FZTST(.RESULT,.FIVE)) THEN
           RESULT(3) = RESULT(3) XOR 080H;
        END;
     ELSE CALL CRT$PRINT$STRING(.('AST$$'));
     CALL SEND$CRLF;
     OK = CHECK$INPUT;
     CALL CLEAR$LOW$SCREEN;
     END;
  END GET$LONG;
```

```
/*****************************************************************************
*
* GET$COURSE$BRG:
*   THIS PROCEDURE IS USED TO OBTAIN THE VALUE OF COURSE OR BEARING FROM
*   THE CRT.
*
* PARAMETERS:
*   - TYPE. - HAS TWO POSSIBLE VALUES:  IF 0 ---> GET COURSE.
*                                       IF 1 ---> GET BEARING.
*   - A. - POINTER TO A MEMORY LOCATION IN WHICH THE FLOATING POINT REPRE-
*     SENTATION OF THE VALUE OF THE COURSE OR BEARING, IN DEGREES, IS DE-
*     SIRED TO BE PLACED.
*
*****************************************************************************/
GET$COURSE$BRG: PROCEDURE (TYPE,A) PUBLIC;
   DCL A ADDRESS,
       RESULT BASED A (4) BYTE,
       BUFFER(7) BYTE,
       (TEMP, OK, TYPE) BYTE;
   OK = 0;
   BUFFER(0) = 4;
   BUFFER(1) = 3;
   BUFFER(6) = 0;
   DO  WHILE OK = 0;
      CALL CRT$PRINT$STRING(.('ENTER THE $$'));
      IF TYPE = 0
         THEN CALL CRT$PRINT$STRING(.('COURSE $$'));
         ELSE CALL CRT$PRINT$STRING(.('BEARING $$'));
      CALL CRT$PRINT$STRING(.('VALUE AS REQUESTED:$$'));
      CALL SEND$CRLF;
      TEMP = 0;
```

```
      DO WHILE TEMP = 0;
         CALL CRT$PRINT$STRING(.('DEGREES:  $$'));
         CALL PUT$NUMBER$BUFFER(3,.BUFFER(2));
         CALL CRT$WRITE(POINT);
         CALL PUT$NUMBER$BUFFER(1,.BUFFER(5));
         CALL ASCII$TO$FLOAT(.BUFFER,7,.RESULT);
         TEMP = CHECK$FP$VALUE(.RESULT,.FP$360);
         END;
      CALL SEND$CRLF;
      OK = CHECK$INPUT;
      CALL CLEAR$LOW$SCREEN;
      END;
   END GET$COURSE$BRG;
```

```
/*****************************************************************************
*
* GET$SPEED:
*   THIS PROCEDURE IS USED TO OBTAIN THE SPEED VALUE FROM THE CRT.
*
* PARAMETERS:
*   - A. - POINTER TO A MEMORY LOCATION IN WHICH THE FLOATING POINT REPRESEN-
*     TATION OF THE VALUE OF THE SPEED IN KNOTS, IS DESIRED TO BE PLACED.
*
*****************************************************************************/
GET$SPEED: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       RESULT BASED A (4) BYTE,
       BUFFER(6) BYTE,
       (TEMP, OK) BYTE;
   OK = 0;
   BUFFER(0) = 3;
   BUFFER(1) = 2;
   BUFFER(5) = 0;
   DO WHILE OK = 0;
      CALL CRT$PRINT$STRING(.('ENTER THE SPEED VALUE AS REQUESTED:$$'));
      CALL SEND$CRLF;
      TEMP = 0;
      DO WHILE TEMP = 0;
         CALL CRT$PRINT$STRING(.('KNOTS:  $$'));
         CALL PUT$NUMBER$BUFFER(2,.BUFFER(2));
         CALL CRT$WRITE(POINT);
         CALL PUT$NUMBER$BUFFER(1,.BUFFER(4));
         CALL ASCII$TO$FLOAT(.BUFFER,6,.RESULT);
         TEMP = CHECK$FP$VALUE(.RESULT,.FP$99$9);
         END;
```

```
      CALL SEND$CRLF;
      OK = CHECK$INPUT;
      CALL CLEAR$LOW$SCREEN;
      END;
   END GET$SPEED;
```

```
/*****************************************************************************
*
* GET$RANGE:
*   THIS PROCEDURE IS USED TO OBTAIN THE FLOATING POINT REPRESENTATION OF
*   A RANGE VALUE OBTAINED FROM THE CRT.
*
* PARAMETERS:
*   - A. - POINTER TO A MEMORY LOCATION IN WHICH THE FLOATING POINT REPRESEN-
*     TATION OF THE VALUE OF A RANGE, IS DESRED TO BE PLACED.
*
* USAGE:
*   ALTHOUGH THIS PROCEDURE WILL ACCEPT EITHER YARDS OR MILES AS UNITS OF
*   RANGE, THE RESULT WILL BE GIVEN IN TERMS OF MILES ONLY.
*
*****************************************************************************/
GET$RANGE: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       RESULT BASED A (4) BYTE,
       BUFFER (8) BYTE,
       (TEMP, OK, CHAR, UNITS) BYTE;
   OK = 0;
   DO WHILE OK = 0;
      CALL CRT$PRINT$STRING(.('ENTER THE RANGE VALUE AS REQUESTED:$$'));
      CALL SEND$CRLF;
      CALL CRT$PRINT$STRING(.('ENTER THE UNITS TO BE USED:  (M/Y)  $$'));
      CHAR = CRT$READ;
      DO WHILE (CHAR <> 'Y') AND (CHAR <> 'M')      /* UPPER CASE CHARACTERS. */
            AND (CHAR <> 'Y') AND (CHAR <> 'M');    /* LOWER CASE CHARACTERS. */
         CALL SEND$BEL;
         CHAR = CRT$READ;
         END;
```

```
IF CHAR >= 61H THEN CHAR = CHAR - 020H;
IF CHAR = 'Y' THEN CALL CRT$PRINT$STRING(.('YARDS. $$'));
                ELSE CALL CRT$PRINT$STRING(.('MILES. $$'));
CALL SEND$CRLF;
TEMP = 0;
DO WHILE TEMP = 0;
   IF CHAR = 'Y'
   THEN DO;
      CALL CRT$PRINT$STRING(.('YARDS:  $$'));
      CALL PUT$NUMBER$BUFFER(6,.BUFFER(2));
      BUFFER(0), BUFFER(1) = 6;
      BUFFER(8) = 0;
      CALL ASCII$TO$FLOAT(.BUFFER,9,.RESULT);
      TEMP = CHECK$FP$VALUE(.RESULT,.FP$202500);
      CALL FDIV(.RESULT,.FP$2000,.RESULT);
      END;
   ELSE DO;
      CALL CRT$PRINT$STRING(.('MILES:  $$'));
      CALL PUT$NUMBER$BUFFER(3,.BUFFER(2));
      CALL CRT$WRITE(POINT);
      CALL PUT$NUMBER$BUFFER(1,.BUFFER(5));
      BUFFER(0) = 4;
      BUFFER(1) = 3;
      BUFFER(6) = 0;
      CALL ASCII$TO$FLOAT(.BUFFER,7,.RESULT);
      TEMP = CHECK$FP$VALUE(.RESULT,.FP$100);
      END;
   END;
CALL SEND$CRLF;
OK = CHECK$INPUT;
CALL CLEAR$LOW$SCREEN;
```

```
    END;
END GET$RANGE;
```

```
/*****************************************************************************
*
* GET$DESIG:
*   THIS PROCEDURE IS USED TO OBTAIN THE DESIGNATION OF THE CONTACT FROM
*   THE CRT.
*
* USAGE:
*   TYPED PROCEDURE.  RETURNS A HASHED VALUE TO BE USED INTERNALLY ACCORDING
*   TO THE FOLLOWING RULE:
*             HASH = CHAR*100 + CHAR1.
*
*****************************************************************************/
GET$DESIG: PROCEDURE ADDRESS PUBLIC;
   DCL HASH ADDRESS,
       BUFFER (2) BYTE,
       (CHAR, OK, CHAR1) BYTE;
   OK = 0;
   DO WHILE OK = 0;
      CALL CRT$PRINT$STRING(.('DESIG:  $$'));
   L: CHAR = CRT$READ;
      DO WHILE ((CHAR < 41H) AND (CHAR <> 20H)) OR
               ((CHAR > 5AH) AND (CHAR < 61H)) OR (CHAR > 7AH);
         CALL SEND$BEL;
         CHAR = CRT$READ;
         END;
      IF CHAR >= 61H THEN CHAR = CHAR - 20H;
      CALL CRT$WRITE(CHAR);
      CHAR1 = CRT$READ;
      DO WHILE ((CHAR1 < 41H) OR ((CHAR1 > 5AH) AND
               (CHAR1 < 61H)) OR (CHAR1 > 7AH)) AND (CHAR1 <> 07FH);
         CALL SEND$BEL;
```

```
        CHAR1 = CRT$READ;
        END;
     IF CHAR1 = 07FH THEN DO;
        CALL SEND$BS;
        GO TO L;
        END;
     IF CHAR1 >= 61H THEN CHAR1 = CHAR1 - 20H;
     CALL CRT$WRITE(CHAR1);
     CALL SEND$CRLF;
     OK = CHECK$INPUT;
     CALL CLEAR$LOW$SCREEN;
     END;
  HASH = CHAR*100 + CHAR1;
  RETURN HASH;
  END GET$DESIG;
```

```
/*****************************************************************************
*
* GET$TYPE:
*    THIS PROCEDURE IS USED TO OBTAIN THE TYPE OF THE CONTACT FROM
*    THE CRT.
*
* USAGE:
*    TYPED PROCEDURE.  RETURNS THE FOLLOWING VALUES :
*                      0 ---> SURFACE.
*                      1 ---> SUB-SURFACE.
*
*****************************************************************************/
GET$TYPE: PROCEDURE BYTE PUBLIC;
   DCL (TYPE, CHAR, OK) BYTE;
   OK = 0;
   DO WHILE OK = 0;
      CALL CRT$PRINT$STRING(.('IS IT A SURFACE TYPE CONTACT?  (Y/N)  $$'));
      CHAR = CHECK$YES$NO;
      CALL SEND$CRLF;
      CALL CRT$PRINT$STRING(.('TYPE:  $$'));
      IF CHAR = 1
      THEN DO;
         CALL CRT$PRINT$STRING(.('SURFACE. $$'));
         TYPE = 0;
         END;
      ELSE DO;
         CALL CRT$PRINT$STRING(.('SUB-SURFACE. $$'));
         TYPE = 1;
         END;
      CALL SEND$CRLF;
      OK = CHECK$INPUT;
```

```
    CALL CLEAR$LOW$SCREEN;
    END;
RETURN TYPE;
END GET$TYPE;
```

```
/*****************************************************************************
*
* GET$KIND:
*   THIS PROCEDURE IS USED TO OBTAIN THE CONTACT CLASS FROM THE CRT.
*
* USAGE:
*   TYPED PROCEDURE.  RETURNS THE FOLLOWING VALUES:
*                     0 ---> FRIEND.
*                     1 ---> HOSTILE.
*                     2 ---> UNKNOWN.
*
*****************************************************************************/
GET$KIND: PROCEDURE BYTE PUBLIC;
   DCL (KIND, TEMP, CHAR, OK) BYTE;
   OK = 0;
   DO WHILE OK = 0;
      TEMP = 0;
      DO WHILE TEMP = 0;
         CALL CRT$PRINT$STRING(.('ENTER THE CONTACT CLASS: (F/H/U)  $$'));
         CALL GET$STRING(.CHAR,1);
         IF (CHAR <> 'F') AND (CHAR <> 'H') AND (CHAR <> 'U')
            THEN CALL PRINT$ERROR$MSG;
            ELSE DO;
                 CALL CRT$WRITE(ETEOL);
                 TEMP = 1;
                 IF CHAR = 'F'
                    THEN DO;
                         CALL CRT$PRINT$STRING(.('RIEND.$$'));
                         KIND = 0;
                         END;
                    ELSE DO;
```

```
                        IF CHAR = 'H'
                           THEN DO;
                                CALL CRT$PRINT$STRING(.('OSTILE.$$'));
                                KIND = 1;
                                END;
                           ELSE DO;
                                CALL CRT$PRINT$STRING(.('NKNOWN.$$'));
                                KIND = 2;
                                END;
                        END;
                  END;
            END;
         CALL SEND$CRLF;
         OK = CHECK$INPUT;
         CALL CLEAR$LOW$SCREEN;
         END;
   RETURN KIND;
   END GET$KIND;
```

```
/*****************************************************************************
*
* GET$SCALE:
*   THIS PROCEDURE IS USED TO OBTAIN A NUMBER REPRESENTING THE SCALE DESIRED
*   TO BE USED IN THE PLOTTING AT THE PLASMA DISPLAY.
*
* PARAMETERS:
*   - A. - POINTER TO A MEMORY LOCATION IN WHICH THE FLOATING POINT REPRESEN-
*     TATION OF THE SCALE (MILES/INCH) IS DESIRED TO BE PLACED.
*
*****************************************************************************/
GET$SCALE: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       SCALE BASED A (4) BYTE,
       BUFFER (7) BYTE,
       (TEMP, TEMP1, OK) BYTE;
   OK = 0;
   DO WHILE OK = 0;
      CALL CRT$PRINT$STRING(.('ENTER THE SCALE AS REQUESTED:$$'));
      CALL SEND$CRLF;
      TEMP, TEMP1 = 0;
      DO WHILE (TEMP = 0) OR (TEMP1 = 0);
         CALL CRT$PRINT$STRING(.('MILES PER INCH: $$'));
         CALL PUT$NUMBER$BUFFER(2,.BUFFER(2));
         CALL CRT$WRITE(POINT);
         CALL PUT$NUMBER$BUFFER(2,.BUFFER(4));
         BUFFER(0) = 4;
         BUFFER(1) = 2;
         BUFFER(6) = 0;
         CALL ASCII$TO$FLOAT(.BUFFER,7,.SCALE);
         TEMP = CHECK$FP$VALUE(.SCALE,.FP$25);
```

```
            IF TEMP <> 0 THEN
               TEMP1 = CHECK$FP$VALUE(.FP$0$25,.SCALE);
            END;
         CALL SEND$CRLF;
         OK = CHECK$INPUT;
         CALL CLEAR$LOW$SCREEN;
         END;
      END GET$SCALE;


END COMMANDS;
```

```
PLASMA$MODULE: DO;

$NOLIST
$INCLUDE (:F1:EXTER.SRC)
$INCLUDE (:F1:EXTER1.SRC)
$INCLUDE (:F1:EXTER2.SRC)
$LIST
```

```
/***   DECLARATIONS:   ***/

DCL TRUE LIT '0FFH',
    FALSE LIT '00H',
    EOL LIT '024H',
    POINT LIT '02EH';

DCL FP$2   (4) BYTE DATA (000H, 000H, 000H, 040H),   /* 2.0 */
    FP$8   (4) BYTE DATA (0CDH, 0CCH, 008H, 041H),   /* 8.55 */
    FP$511 (4) BYTE DATA (000H, 080H, 0FFH, 043H);   /* 511.0 */

DCL LAST$POSI (15) STRUCTURE(
                   FLAG BYTE,
                   X ADDRESS,
                   Y ADDRESS),

    OS$LAST$POSI STRUCTURE(
                 FLAG BYTE,
                 X ADDRESS,
                 Y ADDRESS);

DCL WINDOW (4) BYTE,
    HALF$WINDOW (4) BYTE,
    ORIGIN$X (4) BYTE,
    ORIGIN$Y (4) BYTE;
```

```
DCL BUFFER (*) BYTE INITIAL ('SCALE:        ');
```

```
/*****************************************************************************
*
* SET$WINDOW:
*    THIS PROCEDURE IS USED TO FIND THE VALUES OF THE WINDOW AND HALF$WINDOW
*    FLOATING POINT VECTORS.
*
*****************************************************************************/
SET$WINDOW: PROCEDURE PUBLIC;
    CALL FMUL(.SYSTEM.SCALE, .FP$8, .WINDOW);
    CALL FDIV(.WINDOW, .FP$2, .HALF$WINDOW);
    END SET$WINDOW;
```

```
/*****************************************************************************
*
* CLEAR$STRUCTURES:
*   THIS PROCEDURE IS USED TO CLEAR THE STRUCTURES USED IN THIS MODULE.
*
*****************************************************************************/
CLEAR$STRUCTURES: PROCEDURE PUBLIC;
   DCL A ADDRESS,
       VALUE BASED A BYTE,
       I BYTE;
   A = .LAST$POSI;
   DO I = 0 TO 74;
      VALUE = 00H;
      A = A + 1;
      END;
   A = .OS$LAST$POSI;
   DO I = 0 TO 4;
      VALUE = 00H;
      A = A + 1;
      END;
   END CLEAR$STRUCTURES;
```

```
/*****************************************************************************
*
* DRAW$FRIENDLY$SYMBOL:
*    THIS PROCEDURE IS USED TO DRAW IN THE PLASMA DISPLAY, IN A GIVEN POSITION,
*    THE SYMBOL USED TO REPRESENT A FRIENDLY CONTACT.
*
* PARAMETERS:
*    - X. - ADDRESS VALUE.
*    - Y. - ADDRESS VALUE.
*
*****************************************************************************/
DRAW$FRIENDLY$SYMBOL: PROCEDURE (X, Y) PUBLIC;
   DCL (X, Y) ADDRESS,
       (TEMP$X, TEMP$Y) ADDRESS;
   IF (X < 2) OR (X > 509) OR (Y < 2) OR (Y > 509)
      THEN RETURN;    /* SYMBOL CAN NOT BE DRAWN */
   TEMP$X = X - 1;
   TEMP$Y = Y + 2;
   CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
   TEMP$X = X + 1;
   CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
   TEMP$X = X + 2;
   TEMP$Y = Y + 1;
   CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
   TEMP$Y = Y - 1;
   CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
   TEMP$X = X + 1;
   TEMP$Y = Y - 2;
   CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
   TEMP$X = X - 1;
   CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
```

```
    TEMP$X = X - 2;
    TEMP$Y = Y - 1;
    CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
    TEMP$Y = Y + 1;
    CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
    END DRAW$FRIENDLY$SYMBOL;
```

```
/*****************************************************************************
*
* DRAW$UNK$HOS$SYMBOL:
*    THIS PROCEDURE IS USED TO DRAW IN THE PLASMA DISPLAY, IN A GIVEN POINT,
*    THE SYMBOL USED TO REPRESENT UNKNOWN AND/OR HOSTILE CONTACTS.
*
* PARAMETERS:
*    - X. - ADDRESS VALUE.
*    - Y. - ADDRESS VALUE.
*
*****************************************************************************/
DRAW$UNK$HOS$SYMBOL: PROCEDURE (X, Y) PUBLIC;
   DCL (X, Y, TEMP$X, TEMP$Y) ADDRESS;
   IF (X < 2) OR (X > 509) OR (Y < 2) OR (Y > 509)
      THEN RETURN;    /* SYMBOL CAN NOT BE DRAWN. */
   TEMP$X = X - 2;
   TEMP$Y = Y - 2;
   CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
   TEMP$X = X + 2;
   TEMP$Y = Y + 2;
   CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
   TEMP$X = X - 2;
   CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
   TEMP$X = X + 2;
   TEMP$Y = Y - 2;
   CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
   END DRAW$UNK$HOS$SYMBOL;
```

```
/*****************************************************************************
*
* DRAW$OWN$SHIP$SYMBOL:
*    THIS PROCEDURE IS USED TO DRAW IN THE PLASMA DISPLAY, IN A GIVEN POINT,
*    THE SYMBOL USED TO REPRESENT THE OWN SHIP.
*
* PARAMETERS:
*    - X. - ADDRESS VALUE.
*    - Y. - ADDRESS VALUE.
*
*****************************************************************************/
DRAW$OWN$SHIP$SYMBOL: PROCEDURE (X, Y) PUBLIC;
   DCL (X, Y, TEMP$X, TEMP$Y) ADDRESS;
   CALL DRAW$FRIENDLY$SYMBOL(X, Y);
   IF (X = 0) OR (X = 511) OR (Y = 0) OR (Y = 511)
      THEN RETURN;    /* SYMBOL CAN NOT BE DRAWN. */
   TEMP$X = X - 1;
   TEMP$Y = Y + 1;
   CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
   TEMP$X = X + 1;
   CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
   CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
   TEMP$Y = Y - 1;
   CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
   CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
   TEMP$X = X - 1;
   CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
   CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
   TEMP$Y = Y + 1;
   CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
   END DRAW$OWN$SHIP$SYMBOL;
```

```
/*****************************************************************************
*
* CHECK$PLASMA:
*   THIS PROCEDURE IS USED TO DETERMINE WHETHER OR NOT A POINT REPRESENTED
*   BY TWO X ,Y FP VALUES, CAN BE DISPLAYED AT THE PLASMA.
*
* PARAMETERS:
*   - A. - POINTER TO A FOUR BYTE VECTOR REPRESENTING THE X VALUE;
*   - B. - POINTER TO A FOUR BYTE VECTOR REPRESENTING THE Y VALUE.
*
* USAGE:
*   TYPED PROCEDURE.  RETURNS A VALUE OF 'TRUE' (0FFH) IF THE POINT CAN BE
*   DISPLAYED AT THE CURRENT SCALE; OTHERWISE, A VALUE OF 'FALSE' (00H)
*   IS RETURNED.
*
*****************************************************************************/
CHECK$PLASMA: PROCEDURE (A, B) BYTE PUBLIC;
   DCL (A, B) ADDRESS,
       X BASED A (4) BYTE,
       Y BASED B (4) BYTE,
       TEMP (4) BYTE,
       CHECK BYTE DATA (02H);
   CALL FADD(.ORIGIN$X, .WINDOW, .TEMP);
   IF FCMPR(.X, .TEMP, .CHECK)
      THEN RETURN FALSE;
   IF FCMPR(.ORIGIN$X, .X, .CHECK)
      THEN RETURN FALSE;
   CALL FSUB(.ORIGIN$Y, .WINDOW, .TEMP);
   IF FCMPR(.Y, .ORIGIN$Y, .CHECK)
      THEN RETURN FALSE;
   IF FCMPR(.TEMP, .Y, .CHECK)
```

```
    THEN RETURN FALSE;
    /*  EVERYTHING IS OKAY.  RETURN A TRUE VALUE    */
RETURN TRUE;
END CHECK$PLASMA;
```

```
/*****************************************************************************
*
* NORMALIZE:
*   THIS PROCEDURE IS USED TO NORMALIZE TWO FP VALUES INTO ADDRESS VALUES
*   THAT CAN BE USED TO DETERMINE A POINT IN THE PLASMA DISPLAY.  IT MUST
*   BE USED AFTER 'CHECK$PLASMA'.
*
* PARAMETERS:
*   - A. - POINTER TO A FOUR BYTE VECTOR REPRESENTING A FP 'X' VALUE.
*   - B. - POINTER TO A FOUR BYTE VECTOR REPRESENTING A FP 'Y' VALUE.
*   - C. - POINTER TO AN ADDRESS VALUE IN WHICH THE 'NORMALIZED X' IS
*     DESIRED TO BE PLACED.
*   - D. - POINTER TO AN ADDRESS VALUE IN WHICH THE 'NORMALIZED Y' IS
*     DESIRED TO BE PLACED.
*
*****************************************************************************/
NORMALIZE: PROCEDURE (A, B, C, D) PUBLIC;
   DCL (A, B, C, D) ADDRESS,
       FP$X BASED A (4) BYTE,
       FP$Y BASED B (4) BYTE,
       X BASED C ADDRESS,
       Y BASED D ADDRESS,
       DELTA$X (4) BYTE,
       DELTA$Y (4) BYTE,
       TEMP (4) BYTE,
       TEMP1 (4) BYTE;
   CALL FSUB(.FP$X, .ORIGIN$X, .DELTA$X);
   IF DELTA$X(3) >= 80H
      THEN DELTA$X(3) = DELTA$X(3) XOR 080H;    /* ABSOLUTE VALUE REQUIRED */
   CALL FSUB(.ORIGIN$Y, .FP$Y, .DELTA$Y);
   IF DELTA$Y(3) >= 80H
```

```
    THEN DELTA$Y(3) = DELTA$Y(3) XOR 080H;    /* ABSOLUTE VALUE REQUIRED */
CALL FDIV(.FP$511, .WINDOW, .TEMP);
CALL FMUL(.TEMP, .DELTA$X, .TEMP1);
CALL FIXSD(.TEMP1, .TEMP1);
X = TEMP1(1);
X = SHL(X, 8) + TEMP1(0);
CALL FMUL(.TEMP, .DELTA$Y, .TEMP1);
CALL FIXSD(.TEMP1, .TEMP1);
Y = TEMP1(1);
Y = SHL(Y, 8) + TEMP1(0);
END NORMALIZE;
```

```
/*****************************************************************************
*
* PUT$OS$CENTER:
*    THIS PROCEDURE IS USED TO CHANGE THE X/Y VALUES USED IN THE PLASMA ORI-
*    GIN, IN ORDER TO ALLOW THE OWN SHIP'S LAST POSITION TO BE IN THE NEW
*    PLASMA DISPLAY CENTER.
*
*****************************************************************************/
PUT$OS$CENTER: PROCEDURE PUBLIC;
    DCL POINTER BYTE;
    POINTER = OWN$SHIP$INFO.POINTER;
    CALL FSUB(.OWN$SHIP(POINTER).X, .HALF$WINDOW, .ORIGIN$X);
    CALL FADD(.OWN$SHIP(POINTER).Y, .HALF$WINDOW, .ORIGIN$Y);
    END PUT$OS$CENTER;
```

```
/*****************************************************************************
*
* PUT$CONTACT$CENTER:
*    THIS PROCEDURE IS USED TO CHANGE THE X/Y VALUES USED IN THE PLASMA ORI-
*    GIN, IN ORDER TO ALLOW THE GIVEN CONTACT/LAST POSITION TO BE IN THE NEW
*    PLASMA DISPLAY CENTER.
*
* PARAMETERS:
*    - INDEX. - INDICATES THE RELATIVE POSITION OF THE CONTACT IN THE CONTACT$-
*       INFO STRUCTURE.
*
*****************************************************************************/
PUT$CONTACT$CENTER: PROCEDURE (INDEX) PUBLIC;
    DCL (POINTER, INDEX) BYTE;
    POINTER = CONTACT$INFO(INDEX).POINTER;
    CALL FSUB(.CONTACT$POSI(POINTER).X, .HALF$WINDOW, .ORIGIN$X);
    CALL FADD(.CONTACT$POSI(POINTER).Y, .HALF$WINDOW, .ORIGIN$Y);
    END PUT$CONTACT$CENTER;
```

```
/*****************************************************************************
*
* FIXED$REORIENTATION:
*    THIS PROCEDURE IS USED TO DEFINE A NEW REFERENCE POINT FOR THE PLASMA
*    DISPLAY, ACCORDING TO 8 PREDEFINED POINTS.
*
* PARAMETERS:
*    - TYPE. - BYTE NUMBER BETWEEN 0 AND 7 THAT IS USED TO IDENTIFY THE 8 PRE-
*      DEFINED WAYS TO REORIENT THE PLASMA SCREEN.
*
*****************************************************************************/
FIXED$REORIENTATION: PROCEDURE (TYPE) PUBLIC;
   DCL TYPE BYTE;
   DO CASE TYPE;
      DO;         /* CASE 0 */
        CALL FADD(.ORIGIN$Y, .HALF$WINDOW, .ORIGIN$Y);
      END;

      DO;         /* CASE 1 */
        CALL FADD(.ORIGIN$X, .HALF$WINDOW, .ORIGIN$X);
        CALL FADD(.ORIGIN$Y, .HALF$WINDOW, .ORIGIN$Y);
      END;

      DO;         /* CASE 2 */
        CALL FADD(.ORIGIN$X, .HALF$WINDOW, .ORIGIN$X);
      END;

      DO;         /* CASE 3 */
        CALL FADD(.ORIGIN$X, .HALF$WINDOW, .ORIGIN$X);
        CALL FSUB(.ORIGIN$Y, .HALF$WINDOW, .ORIGIN$Y);
      END;
```

```
      DO;         /* CASE 4 */
        CALL FSUB(.ORIGIN$Y, .HALF$WINDOW, .ORIGIN$Y);
      END;

      DO;         /* CASE 5 */
        CALL FSUB(.ORIGIN$X, .HALF$WINDOW, .ORIGIN$X);
        CALL FSUB(.ORIGIN$Y, .HALF$WINDOW, .ORIGIN$Y);
      END;

      DO;         /* CASE 6 */
        CALL FSUB (.ORIGIN$X, .HALF$WINDOW, .ORIGIN$X);
      END;

      DO;         /* CASE 7 */
        CALL FSUB(.ORIGIN$X, .HALF$WINDOW, .ORIGIN$X);
        CALL FADD(.ORIGIN$Y, .HALF$WINDOW, .ORIGIN$Y);
      END;

    END;      /* END CASE */
  END FIXED$REORIENTATION;
```

```
/*****************************************************************************
*
* PLASMA$REDESIG:
*    THIS PROCEDURE IS USED TO DISPLAY, IF POSSIBLE, THE NEW CONTACT'S DESIG
*    IN THE LAST KNOWN POSITION.
*
* PARAMETERS:
*    - INDEX. - INDICATES THE RELATIVE POSITION OF THE CONTACT IN THE CONTACT$-
*       INFO STRUCTURE.
*
*****************************************************************************/
PLASMA$REDESIG: PROCEDURE (INDEX) PUBLIC;
   DCL (X, Y) ADDRESS,
       INDEX BYTE;
   IF LAST$POSI(INDEX).FLAG
   THEN DO;
        X = LAST$POSI(INDEX).X;
        Y = LAST$POSI(INDEX).Y;
        CALL GRAPHIC$DESIG(X, Y, .CONTACT$INFO(INDEX).DESIG);
        END;
   END PLASMA$REDESIG;
```

```
/*****************************************************************************
*
* PLASMA$DELETE:
*   THISPROCEDURE IS USED TO CLEAR THE FLAG CORRESPONDING TO A CONTACT THAT
*   HAS BEEN REMOVED FROM THE SYSTEM.
*
* PARAMETERS:
*   - INDEX. - INDICATES THE RELATIVE POSITION OF THE CONTACT IN THE CONTACT$-
*     INFO STRUCTURE.
*
*****************************************************************************/
PLASMA$DELETE: PROCEDURE (INDEX) PUBLIC;
   DCL INDEX BYTE;
   LAST$POSI(INDEX).FLAG = FALSE;
   END PLASMA$DELETE;
```

```
/*****************************************************************************
*
* PLASMA$CONTACT:
*   THIS PROCEDURE IS USED TO TRY TO PLOT THE LAST POSITION OF A GIVEN CONT-
*   ACT, IF POSSIBLE.
*
* PARAMETERS:
*   - INDEX. - INDICATES THE RELATIVE POSITION OF THE CONTACT IN THE CONTACT$-
*     INFO STRUCTURE.
*
*****************************************************************************/
PLASMA$CONTACT: PROCEDURE (INDEX) PUBLIC;
   DCL (X, Y) ADDRESS,
       (POINTER, INDEX, TEMP) BYTE;
   POINTER = CONTACT$INFO(INDEX).POINTER;
   TEMP = CHECK$PLASMA(.CONTACT$POSI(POINTER).X, .CONTACT$POSI(POINTER).Y);
   IF LAST$POSI(INDEX).FLAG
   THEN DO;
        IF TEMP
        THEN DO;
             CALL START$VECTOR$DASH(LAST$POSI(INDEX).X, LAST$POSI(INDEX).Y);
             CALL NORMALIZE(.CONTACT$POSI(POINTER).X, .CONTACT$POSI(POINTER).Y,
                            .X, .Y);
             CALL STOP$VECTOR$DASH(X, Y);
             IF CONTACT$INFO(INDEX).KIND = 0
                THEN CALL DRAW$FRIENDLY$SYMBOL(X, Y);
                ELSE CALL DRAW$UNK$HOS$SYMBOL(X, Y);
             LAST$POSI(INDEX).X = X;
             LAST$POSI(INDEX).Y = Y;
             END;
        ELSE DO;
```

```
            LAST$POSI(INDEX).FLAG = FALSE;
            END;
      END;
   ELSE DO;
      IF TEMP
      THEN DO;
            CALL NORMALIZE(.CONTACT$POSI(POINTER).X, .CONTACT$POSI(POINTER).Y,
                           .X, .Y);
            CALL GRAPHIC$DESIG(X, Y, .CONTACT$INFO(INDEX).DESIG);
            IF CONTACT$INFO(INDEX).KIND = 0
               THEN CALL DRAW$FRIENDLY$SYMBOL(X, Y);
               ELSE CALL DRAW$UNK$HOS$SYMBOL(X, Y);
            LAST$POSI(INDEX).FLAG = TRUE;
            LAST$POSI(INDEX).X = X;
            LAST$POSI(INDEX).Y = Y;
            END;
      END;
END PLASMA$CONTACT;
```

```
/*****************************************************************************
*
* PLASMA$OS:
*    THIS PROCEDURE IS USED TO DISPLAY, IF POSSIBLE, THE LAST KNOWN POSITION
*    OF THE OWN SHIP IN THE PLASMA DISPLAY.
*
*****************************************************************************/
PLASMA$OS: PROCEDURE PUBLIC;
    DCL (X, Y) ADDRESS,
        (POINTER, TEMP) BYTE;
    POINTER = OWN$SHIP$INFO.POINTER;
    TEMP = CHECK$PLASMA(.OWN$SHIP(POINTER).X, .OWN$SHIP(POINTER).Y);
    IF OS$LAST$POSI.FLAG
    THEN DO;
         IF TEMP
         THEN DO;
              CALL START$VECTOR$SOLID(OS$LAST$POSI.X, OS$LAST$POSI.Y);
              CALL NORMALIZE(.OWN$SHIP(POINTER).X, .OWN$SHIP(POINTER).Y,
                             .X, .Y);
              CALL STOP$VECTOR$SOLID(X, Y);
              CALL DRAW$OWN$SHIP$SYMBOL(X, Y);
              OS$LAST$POSI.X = X;
              OS$LAST$POSI.Y = Y;
              END;
         ELSE DO;
              OS$LAST$POSI.FLAG = FALSE;
              END;
         END;
    ELSE DO;
         IF TEMP
         THEN DO;
```

```
            CALL NORMALIZE(.OWN$SHIP(POINTER).X, .OWN$SHIP(POINTER).Y,
                           .X, .Y);
            CALL DRAW$OWN$SHIP$SYMBOL(X, Y);
            OS$LAST$POSI.FLAG = TRUE;
            OS$LAST$POSI.X = X;
            OS$LAST$POSI.Y = Y;
            END;
      END;
END PLASMA$OS;
```

```
/*****************************************************************************
*
* DRAW$EVERYTHING:
*   THIS PROCEDURE IS USED TO REDRAW THE ENTIRE DISPLAY.
*
*****************************************************************************/
DRAW$EVERYTHING: PROCEDURE PUBLIC;
   DCL (POINTER, I, J ,P, TEMP) BYTE;
   CALL CLEAR$PLASMA;
   POINTER = OWN$SHIP$INFO.POINTER;
   IF OWN$SHIP$INFO.FLAG
   THEN DO;
        P = POINTER + 1;
        DO I = 0 TO 29;
           IF P >= 30 THEN P = 0;
           OWN$SHIP$INFO.POINTER = P;
           CALL PLASMA$OS;
           P = P + 1;
           END;
        END;
   ELSE DO;
        DO I = 0 TO POINTER;
           OWN$SHIP$INFO.POINTER = I;
           CALL PLASMA$OS;
           END;
        END;
   OWN$SHIP$INFO.POINTER = POINTER;
   DO I = 0 TO 14;
      IF CONTACT$INFO(I).DESIG <> 00H
      THEN DO;
           POINTER = CONTACT$INFO(I).POINTER;
```

```
            TEMP = CONTACT$INFO(I).POINTER / 15;
            TEMP = (TEMP + 1) * 15;
            IF CONTACT$INFO(I).FLAG
            THEN DO;
                 P = POINTER + 1;
                 DO J = 0 TO 14;
                    IF P >= TEMP THEN P = (POINTER / 15) * 15;
                    CONTACT$INFO(I).POINTER = P;
                    CALL PLASMA$CONTACT(I);
                    P = P + 1;
                    END;
                 END;
            ELSE DO;
                 DO J = ((POINTER / 15) * 15) TO POINTER;
                    CONTACT$INFO(I).POINTER = J;
                    CALL PLASMA$CONTACT(I);
                    END;
                 END;
            CONTACT$INFO(I).POINTER = POINTER;
            END;
      END;
   END DRAW$EVERYTHING;
```

```
/*****************************************************************************
*
* DISPLAY$PLASMA$SCALE:
*   THIS PROCEDURE IS USED TO DISPLAY AT THE PLASMA DISPLAY, THE SCALE BEING
*   USED TO DRAW THE PICTURE.
*
*****************************************************************************/
DISPLAY$PLASMA$SCALE: PROCEDURE PUBLIC;
   DCL TEMP BYTE;
   TEMP = FP$FORMAT(.SYSTEM.SCALE, .BUFFER(7), 2, 2);
   BUFFER(12), BUFFER(13) = EOL;
   BUFFER(11) = BUFFER(10);
   BUFFER(10) = BUFFER(9);
   BUFFER(9) = POINT;
   CALL PLASMA$PRINT$STRING(0, 2, .BUFFER);
   END DISPLAY$PLASMA$SCALE;
```

```
/*****************************************************************************
*
* REORIENT$PS:
*   THIS PROCEDURE IS USED TO DEFINE A NEW REFERENCE POINT FOR THE PICTURE
*   TO BE PRESENTED AT THE PLASMA DISPLAY.
*
*****************************************************************************/
REORIENT$PS: PROCEDURE PUBLIC;
   DCL DESIG ADDRESS,
       (TYPE, OK, REORIENT, INDEX, TEMP, FLAG, COUNT) BYTE;
   DCL TITLE (*) BYTE DATA
             ('                                    PICTURE REORIENTATION.$$'),
       MSG0 (*) BYTE DATA
            ('IF FIXED REORIENTATION IS DESIRED, TYPE 0.$$'),
       MSG1 (*) BYTE DATA
            ('IF OWN SHIP AT CENTER IS DESIRED, TYPE 1.$$'),
       MSG2 (*) BYTE DATA
            ('IF A CONTACT IS DESIRED AT CENTER, TYPE 2.$$'),
       MSG3 (*) BYTE DATA
            ('ENTER VALUE:  $$'),
       MSG4 (*) BYTE DATA
            ('ENTER POINT DESIRED TO BE AT CENTER:  $$'),
       MSG5 (*) BYTE DATA
            ('ENTER CONTACT DESIG AS REQUESTED:$$'),
       MSG6 (*) BYTE DATA
            ('DESIG NOT IN USE.$$');

   IF SYSTEM.NUMCTS > 0
   THEN DO;
        FLAG = TRUE;
        COUNT = '2';
```

```
      END;
ELSE DO;
      FLAG = FALSE;
      COUNT = '1';
      END;
CALL CLEAR$STRUCTURES;
CALL SET$WINDOW;
OK = 0;
DO WHILE OK = 0;
   CALL CRT$PRINT$STRING(.TITLE);
   CALL SEND$CRLF;
   CALL CRT$PRINT$STRING(.MSG0);
   CALL SEND$CRLF;
   CALL CRT$PRINT$STRING(.MSG1);
   CALL SEND$CRLF;
   IF FLAG
   THEN DO;
      CALL CRT$PRINT$STRING(.MSG2);
      CALL SEND$CRLF;
      END;
   CALL CRT$PRINT$STRING(.MSG3);
   REORIENT = CRT$READ;
   DO WHILE (REORIENT < '0') OR (REORIENT > COUNT);
      CALL SEND$BEL;
      REORIENT = CRT$READ;
      END;
   CALL CRT$WRITE(REORIENT);
   CALL SEND$CRLF;
   OK = CHECK$INPUT;
   CALL CLEAR$LOW$SCREEN;
   END;
```

```
REORIENT = REORIENT - 30H;
DO CASE REORIENT;

   DO;       /* CASE 0.  FIXED REORIENTATION.  */
      OK = 0;
      DO WHILE OK = 0;
         CALL CRT$PRINT$STRING(.TITLE);
         CALL SEND$CRLF;
         CALL CRT$PRINT$STRING(.MSG4);
         TYPE = CRT$READ;
         DO WHILE (TYPE < '0') OR (TYPE > '7');
            CALL SEND$BEL;
            TYPE = CRT$READ;
            END;
         CALL CRT$WRITE(TYPE);
         CALL SEND$CRLF;
         OK = CHECK$INPUT;
         CALL CLEAR$LOW$SCREEN;
         END;
      CALL FIXED$REORIENTATION(TYPE - 30H);
   END;
   DO;       /* CASE 1.  OWN SHIP AT CENTER.  */
      CALL PUT$OS$CENTER;
   END;
   DO;       /* CASE 2.  CONTACT AT CENTER.  */
      OK = 0FFH;
      DO WHILE OK = 0FFH;
         CALL CRT$PRINT$STRING(.TITLE);
         CALL SEND$CRLF;
         CALL CRT$PRINT$STRING(.MSG5);
         CALL SEND$CRLF;
```

```
            DESIG = GET$DESIG;
            INDEX = CHECK$DESIG(DESIG);
            IF INDEX = 0FFH
            THEN DO;
                 CALL CRT$PRINT$STRING(.MSG6);
                 CALL SEND$CRLF;
                 CALL SEND$CRLF;
                 CALL CHECK$GO$KEY;
                 END;
            OK = INDEX;
            CALL CLEAR$LOW$SCREEN;
            END;
         CALL PUT$CONTACT$CENTER(INDEX);
      END;

      END;       /* END CASE */

   CALL DRAW$EVERYTHING;
   CALL DISPLAY$PLASMA$SCALE;
   END REORIENT$PS;


END PLASMA$MODULE;
```

```
CRT: DO;


CRT$WRITE:
   PROCEDURE (A) EXTERNAL;
   DECLARE A BYTE; END;

CRT$PRINT$STRING:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

DECLARE
   LIT LITERALLY 'LITERALLY',
   DCL LIT 'DECLARE';

DCL LF LIT '0AH',                       /* LINE FEED.  */
    MASTER$CLEAR LIT '1EH',             /* MASTER CLEAR.  */
    BLINK LIT '0EH',                    /* BLINK ON.  */
    ETEOL LIT '17H',                    /* ERASE TO END OF LINE.  */
    EOL LIT '24H',                      /* END OF LINE.  */
    HOME LIT '02H',                     /* HOME.  */
    TAB LIT '09H',                      /* TAB.  */
    FS LIT '1CH',                       /* FORWARD CURSOR. NON-DESTRUCTIVE.  */
    PROT$FIELD LIT '0FH',               /* START PROTECTED FIELD.  */
    END$PF LIT '18H',                   /* STOP PROTECTED FIELD.  */
    SPACE LIT '20H';                    /* SPACE.  */


DCL LIN1 (*) BYTE DATA
         ('TIME: ($3) \%5\%4\%5\%4LAST  MARK%4\%6\%5\%6C P AM7#'),
```

```
LIN2A (*) BYTE DATA
      ('$2:$2:$2%4\DESIG\TYPE\CLASS\ TIME\ BRG \ RNG  \#'),
LIN2B (*) BYTE DATA
       ('COURSE\SPEED\ TIME\ BRG \ RNG  #'),
LIN3A (*) BYTE DATA
      ('%C#'),
LIN4A (*) BYTE DATA
      ('LAT:%8#'),
LIN5A (*) BYTE DATA
      ('$3:$2. $1  $1 #'),
LIN6A (*) BYTE DATA
      ('LONG:%7#'),
LIN9A (*) BYTE DATA
      ('COURSE:$3. $1#'),
LIN10A (*) BYTE DATA
      ('SPEED:  $2. $1#'),
LIN12A (*) BYTE DATA
      ('CONTACTS:   #'),
LIN13A (*) BYTE DATA
      (' $2F $2H $2U#'),
LIN15A (*) BYTE DATA
      ('MODE:$7#'),
LIN16 (*) BYTE DATA
      ('%%%%#'),
LINF (*) BYTE DATA
      ('\%5\%4\%5\%5\%5\%6\%6\%5\%5\%5\%6#'),
LINE (*) BYTE DATA
      ('\  $2 \ $2 \ $3 \$2:$2\$3. $1\$6\ $3. $1\ $2. $1\$2:$2\$3. $1\$6#');
```

```
/*****************************************************************************
*
* CRT$MASTER$CLEAR:
*    THIS PROCEDURE WILL CLEAR THE ENTIRE SCREEN, AND PUT THE CURSOR AT HOME.
*
*****************************************************************************/
CRT$MASTER$CLEAR: PROCEDURE PUBLIC;
      CALL CRT$WRITE(MASTER$CLEAR);
                END CRT$MASTER$CLEAR;
```

```
/*****************************************************************************
*
* SET$LOW$HOME:
*    THIS PROCEDURE WILL LOCATE THE CURSOR AT THE FIRST COLUMN IN ROW 17.
*
*****************************************************************************/
SET$LOW$HOME: PROCEDURE PUBLIC;
   DCL I BYTE;
   CALL CRT$WRITE(HOME);
   DO I = 1 TO 8;
      CALL CRT$WRITE(LF);
      END;
   END SET$LOW$HOME;
```

```
/*****************************************************************************
*
* CLEAR$LOW$SCREEN:
*    THIS PROCEDURE WILL CLEAR ROWS 17 THRU 24.
*    AFTER THIS OPERATION, THE CURSOR WILL BE PLACED AT COLUMN 1 IN
*    ROW 17.
                                    *
*****************************************************************************/
CLEAR$LOW$SCREEN: PROCEDURE PUBLIC;
   DCL I BYTE;
   CALL SET$LOW$HOME;
   CALL CRT$WRITE(ETEOL);
   DO I = 1 TO 7;
      CALL CRT$WRITE(LF);
      CALL CRT$WRITE(ETEOL);
       END;
                  CALL SET$LOW$HOME;
   END CLEAR$LOW$SCREEN;
```

```
/*****************************************************************************
*
* SET$HIGH$HOME:
*    THIS PROCEDURE WILL LOCATE THE CURSOR AT THE COLUMN 1 AT ROW 1.
*
*****************************************************************************/
SET$HIGH$HOME: PROCEDURE PUBLIC;
    CALL CRT$WRITE(HOME);
    END SET$HIGH$HOME;
```

```
/*****************************************************************************
*                                                                  * PUT$SPACE:
 *    THIS PROCEDURE WILL WRITE A CERTAIN NUMBER OF SPACES AT THE CRT.
      *   :
      * PARAMETERS:
*     - NUM. - NUMBER OF SPACES TO BE WRITTEN.
 *
 *****************************************************************************/
PUT$SPACE: PROCEDURE(NUM);
    DCL NUM BYTE;
     DO WHILE NUM > 0;
        CALL CRT$WRITE(SPACE);
          NUM = NUM - 1;
        END;
    END PUT$SPACE;
```

```
/***** ** ***********************************************************************
                *
     :          * PUT$TAB:
 *                 THIS PROCEDURE WILL SEND A GIVEN NUMBER OF 'TABS' TO THE CRT.
*
            * PARAMETERS:
                *    - NUM. - NUMBER OF TABS DESIRED.
*
***********************************************************************/
PUT$TAB: PROCEDURE(NUM);
    DCL NUM BYTE;
    DO WHILE NUM > 0;
       CALL CRT$WRITE(TAB);
       NUM = NUM - 1;
       END;
    END PUT$TAB;
```

```
/*****************************************************************************
*
* PUT$FS:
*   THIS PROCEDURE WILL SEND TO THE CRT A GIVEN NUMBER OF NON-DESTRUCTIVE
*   FORWARD CURSOR CHARACTERS.
*
* PARAMETERS:
*   - NUM. - NUMBER OF NON-DESTRUCTIVE FORWARD CURSOR CHARACTERS DESIRED.
*
*****************************************************************************/
PUT$FS:  PROCEDURE (NUM);
    DCL NUM BYTE;
     DO WHILE NUM > 0;
            CALL CRT$WRITE(FS);
       NUM = NUM - 1;
       END;
    END PUT$FS;
```

```
/*****************************************************************************
*
* PUT$LF:
*    THIS PROCEDURE WILL SEND AN SPECIFIED NUMBER OF LINE FEED CHARACTERS
*    TO THE CRT.
*
                                        * PARAMETERS:
*     - NUM. - NUMBER OF LINE FEED CHARACTERS DESIRED.
      *
*****************************************************************************/
PUT$LF: PROCEDURE(NUM);
    DCL NUM BYTE;
    DO WHILE NUM > 0;
       CALL CRT$WRITE(LF);
       NUM = NUM - 1;
       END;
    END PUT$LF;
```

```
/*****************************************************************************
*
* START$PROT$FIELD:
*   THIS PROCEDURE WILL CAUSE ALL CHARACTERS SENT AFTER IT FINISHES, TO
*   BE IN A PROTECTED FIELD.
*
*****************************************************************************/
START$PROT$FIELD: PROCEDURE;
   CALL CRT$WRITE(PROT$FIELD);
   END START$PROT$FIELD;
```

```
/*****************************************************************************
*
* START$BLINK:
*   THIS PROCEDURE WILL CAUSE ALL CHARACTERS SENT AFTER IT TO BLINK.
*
*****************************************************************************/
START$BLINK: PROCEDURE PUBLIC;
   CALL CRT$WRITE(BLINK);
   END START$BLINK;
```

```
/*****************************************************************************
*
* STOP$PROT$FIELD:
*   THIS PROCEDURE WILL CAUSE THE END TO 'INSERT PROTECTED FIELD',
*   'ROLL MODE' AND 'BLINK ON'.
*
*****************************************************************************/
STOP$PROT$FIELD: PROCEDURE;
   CALL CRT$WRITE(END$PF);
   END STOP$PROT$FIELD;
```

```
/*****************************************************************************
*
* INTERP:
*   THIS PROCEDURE IS USED BY 'INIT$HIGH$SCREEN' TO INTERPRET THE
*   STREAM OF CHARACTERS USED TO INITIALIZE THE HIGH PORTION OF THE
*   SCREEN.  (ROWS 1 THRU 16)
*
* PARAMETERS:
*   - A. - POINTER TO THE FIRST LOCATION OF A STRING OF CONTROL CHARACTERS
*          USED TO FORMAT THE ROWS ON THE HIGH PORTION OF THE SCREEN.
*
*****************************************************************************/
INTERP: PROCEDURE (A);
   DCL A ADDRESS,
       ARRAY BASED A (80) BYTE,
       (I,NUM) BYTE;
   CALL START$PROT$FIELD;
   I = 0;
   DO WHILE ARRAY(I) <> '#';
      IF ARRAY(I) = '$'
      THEN DO;
         CALL STOP$PROT$FIELD;
         I = I + 1;
         NUM = ARRAY(I) - 30H;
         CALL PUT$SPACE(NUM);
         CALL START$PROT$FIELD;
         END;
      ELSE DO;
         IF ARRAY(I) = '%'
         THEN DO;
            I = I + 1;
```

```
            NUM = ARRAY(I) - 30H;
            CALL PUT$SPACE(NUM);
            END;
         ELSE DO;
            CALL CRT$WRITE(ARRAY(I));
            END;
         END;
      I = I + 1;
      END;
   CALL STOP$PROT$FIELD;
   END INTERP;
```

```
/*****************************************************************************
*
* INIT$HIGH$SCREEN:
*   THIS PROCEDURE IS USED TO INITIALIZE THE HIGH PORTION OF THE
*   SCREEN, WITH A FORMAT PRE-STABLISHED.
*
*****************************************************************************/
INIT$HIGH$SCREEN: PROCEDURE PUBLIC;
   CALL STOP$PROT$FIELD;                    /* TO END ROLL MODE, IF SET. */
   CALL SET$HIGH$HOME;
   CALL INTERP(.LIN1);                      /* LINE 1. */
   CALL INTERP(.LIN2A);                     /* LINE 2. */
   CALL INTERP(.LIN2B);
   CALL INTERP(.LIN3A);                     /* LINE 3. */
   CALL INTERP(.LINF);
   CALL INTERP(.LIN4A);                     /* LINE 4. */
   CALL INTERP(.LINE);
   CALL INTERP(.LIN5A);                     /* LINE 5. */
   CALL INTERP(.LINF);
   CALL INTERP(.LIN6A);                     /* LINE 6. */
   CALL INTERP(.LINE);
   CALL INTERP(.LIN5A);                     /* LINE 7. */
   CALL INTERP(.LINF);
   CALL INTERP(.LIN3A);                     /* LINE 8. */
   CALL INTERP(.LINE);
   CALL INTERP(.LIN9A);                     /* LINE 9. */
   CALL INTERP(.LINF);
   CALL INTERP(.LIN10A);                    /* LINE 10. */
   CALL INTERP(.LINE);
   CALL INTERP(.LIN3A);                     /* LINE 11. */
   CALL INTERP(.LINF);
```

```
   CALL INTERP(.LIN12A);                    /* LINE 12.  */
   CALL INTERP(.LINE);
   CALL INTERP(.LIN13A);                    /* LINE 13.  */
   CALL INTERP(.LINF);
   CALL INTERP(.LIN3A);                     /* LINE 14.  */
   CALL INTERP(.LINE);
   CALL INTERP(.LIN15A);                    /* LINE 15.  */
   CALL INTERP(.LINF);
   CALL INTERP(.LIN16);                     /* LINE 16.  */
   END INIT$HIGH$SCREEN;
```

```
/*****************************************************************************
*
* PRINT$TIME$ZONE:
*   THIS PROCEDURE WILL PRINT THE CURRENT TIME ZONE NUMBER.
*
* PARAMETERS:
*   - A. - POINTER TO A STRING CONTAINING THE ASCII CHARACTERS REPRESENTING
*     THE TIME ZONE NUMBER.
*
*****************************************************************************/
PRINT$TIME$ZONE: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       BUFFER BASED A (5) BYTE;
   CALL SET$HIGH$HOME;
   CALL CRT$PRINT$STRING(.BUFFER);
   CALL SET$LOW$HOME;
   END PRINT$TIME$ZONE;
```

```
/*****************************************************************************
*
* PRINT$TIME:
*   THIS PROCEDURE WILL PRINT THE LOCAL TIME AT THE CRT.
*
* PARAMETERS:
*   - A. - POINTER TO A STRING OF ASCII CHARACTERS REPRESENTING THE TIME.
*
*****************************************************************************/
PRINT$TIME: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       I BYTE,
       BUFFER BASED A (8) BYTE;
   CALL SET$HIGH$HOME;
   CALL PUT$TAB(1);
   DO I = 0 TO 5;
      CALL CRT$WRITE(BUFFER(I));
      END;
   CALL SET$LOW$HOME;
   END PRINT$TIME;
```

```
/*****************************************************************************
*
* PRINT$LAT$LONG:
*   THIS PROCEDURE WILL DISPLAY THE CURRENT LATITUDE AND LONGITUDE AT THE
*   CRT.
*
* PARAMETERS:
*   - A. - POINTER TO A STRING OF ASCII CHARACTERS REPRESENTING THE LATITUDE.
*   - B. - POINTER TO A STRING OF ASCII CHARACTERS REPRESENTING THE LONGITUDE.
*
*****************************************************************************/
PRINT$LAT$LONG: PROCEDURE (A,B) PUBLIC;
   DCL (A,B)ADDRESS,
       BUFF1 BASED A (10) BYTE,
       BUFF2 BASED B (10) BYTE;
   CALL SET$HIGH$HOME;
   CALL PUT$LF(2);
   CALL PUT$TAB(16);
   CALL PUT$FS(6);
   CALL CRT$PRINT$STRING(.BUFF1);
   CALL SET$HIGH$HOME;
   CALL PUT$LF(3);
   CALL PUT$TAB(16);
   CALL PUT$FS(6);
   CALL CRT$PRINT$STRING(.BUFF2);
   CALL SET$LOW$HOME;
   END PRINT$LAT$LONG;
```

```
/*****************************************************************************
*
* PRINT$COURSE:
*   THIS PROCEDURE WILL PRINT THE CURRENT OWN COURSE VALUE AT THE
*   CRT.
*
* PARAMETERS:
*   - A. - POINTER TO A STRING OF ASCII CHARACTERS REPRESENTING THE COURSE.
*
*****************************************************************************/
PRINT$COURSE: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       BUFFER BASED A (6) BYTE;
   CALL SET$HIGH$HOME;
   CALL PUT$LF(4);
   CALL PUT$TAB(17);
   CALL CRT$PRINT$STRING(.BUFFER);
   CALL SET$LOW$HOME;
   END PRINT$COURSE;
```

```
/*****************************************************************************
*
* PRINT$SPEED:
*   THIS PROCEDURE WILL PRINT THE CURRENT OWN SPEED AT THE CRT.
*
* PARAMETERS:
*   - A. - POINTER TO A STRING OF ASCII CHARACTERS REPRESENTING THE SPEED.
*
*****************************************************************************/
PRINT$SPEED: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       BUFFER BASED A (5) BYTE;
   CALL SET$HIGH$HOME;
   CALL PUT$LF(5);
   CALL CRT$PRINT$STRING(.BUFFER);
   CALL SET$LOW$HOME;
   END PRINT$SPEED;
```

```
/*****************************************************************************
*
* PRINT$CONTACTS:
*   THIS PROCEDURE WILL PRINT THE NUMBER OF FRIEND, HOSTILE AND UNKNOWN
*   CONTACTS BEING PROCESSED BY THE SYSTEM.
*
* PARAMETERS:
*   - A. - POINTER TO A STRING OF ASCII CHARACTERS REPRESENTING THE NUMBER
*     OF FRIEND, HOSTILE AND UNKNOWN CONTACTS.
*
*****************************************************************************/
PRINT$CONTACTS: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       BUFFER BASED A (8) BYTE;
   CALL SET$HIGH$HOME;
   CALL PUT$LF(6);
   CALL PUT$TAB(17);
   CALL CRT$PRINT$STRING(.BUFFER);
   CALL SET$LOW$HOME;
   END PRINT$CONTACTS;
```

```
/*****************************************************************************
*
* PRINT$MODE:
*   THIS PROCEDURE WILL PRINT THE CURRENT OPERATING MODE.
*
* PARAMETERS:
*   - A. - POINTER TO A STRING OF ASCII CHARACTERS DEFINING THE CURRENT
*     OPERATING MODE.
*
*****************************************************************************/
PRINT$MODE: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       BUFFER BASED A (9) BYTE;
   CALL SET$HIGH$HOME;
   CALL PUT$LF(7);
   CALL PUT$TAB(17);
   CALL CRT$PRINT$STRING(.BUFFER);
   CALL SET$LOW$HOME;
   END PRINT$MODE;
```

```
/*****************************************************************************
*
* PRINT$CONTACT$INFO:
*   THIS PROCEDURE WILL PRINT ALL THE CURRENT INFORMATION OF ANY CONTACT
*   BEING DISPLAYED.
*
* PARAMETERS:
*   - NUM. - REPRESENTS THE RELATIVE POSITION FROM TOP TO BOTTOM. OF THE
*     CONTACT LINE DESIRED TO BE UPDATED.
*   - A. - POINTER TO A STRING OF ASCII CHARACTERS REPRESENTING THE CONTACT
*     INFORMATION TO BE DISPLAYED.
*
*****************************************************************************/
PRINT$CONTACT$INFO: PROCEDURE (NUM,A) PUBLIC;
   DCL A ADDRESS,
       BUFFER BASED A (44) BYTE,
       NUM BYTE;
   CALL SET$HIGH$HOME;
   CALL PUT$LF(NUM + 1);
   IF NUM = 4 THEN CALL PUT$TAB(2);
   CALL CRT$PRINT$STRING(.BUFFER);
   CALL SET$LOW$HOME;
   END PRINT$CONTACT$INFO;


END CRT;
```

FLTASCII

FLTASCII: DO;

```
/***   EXTERNALS:   ***/

EDIV:
   PROCEDURE (A,B,C,D) EXTERNAL;
   DECLARE (A,B,C,D) ADDRESS; END;

FMUL:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B,C) ADDRESS; END;

FDIV:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B,C) ADDRESS; END;

FADD:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B,C) ADDRESS; END;

FSUB:
   PROCEDURE (A,B,C) EXTERNAL;
   DECLARE (A,B,C) ADDRESS; END;

FLTDS:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE (A,B) ADDRESS; END;

FIXSD:
   PROCEDURE (A,B) EXTERNAL;
   DECLARE (A,B) ADDRESS; END;
```

```
FZTST:
   PROCEDURE (A,B) BYTE EXTERNAL;
   DECLARE (A,B) ADDRESS; END;

DECLARE LIT LITERALLY 'LITERALLY',
        DCL LIT 'DECLARE';
```

```
/*****************************************************************************
*
* ASCII$TO$FLOAT:
*   PROCEDURE USED TO CONVERT A STRING OF ASCII CHARACTERS INTO A FLOATING
*   POINT NUMBER, ACCORDING TO THE FORMAT REQUIRED TO OPERATE ON THE F.P.
*   BOARD.
*
* PARAMETERS:
*   - A. - POINTER TO A N BYTE VECTOR CONTAINING THE ASCII STRING OF NUMBERS
*     REPRESENTING A DECIMAL VALUE DESIRED TO BE REPRESENTED IN FLOATING
*     POINT FORMAT.  IF THE N BYTE VECTOR IS REPRESENTED BY THE
*     NAME 'BUFFER', THEN THE FOLLOWING RULES APPLY:
*         BUFFER(0). - CONTAINS THE DECIMAL NUMBER OF ASCII CHARACTERS PRE-
*                      SENT IN THE BUFFER.  A VALUE OF 0 WILL DENOTE THE NUM-
*                      BER 0.0.
*         BUFFER(1). - CONTAINS THE DECIMAL NUMBER OF ASCII CHARACTERS THAT
*                      REPRESENT THE INTEGER PORTION OF THE NUMBER.  A VALUE OF
*                      0 WILL MEAN THAT THE NUMBER IS LESS THAN ONE.
*         BUFFER(2) --> BUFFER(N-2). - EACH BYTE CONTAINS AN ASCII CHA-
*                      RACTER.  (HEXADECIMAL VALUE)
*         BUFFER(N-1). - A VALUE OF DECIMAL 1 WILL MEAN THAT THE NUMBER IS NE-
*                      GATIVE.
*
*   - LIMIT. - TOTAL NUMBER OF BYTES IN 'BUFFER'.  (N)
*
*
*   - B. - POINTER TO A FOUR BYTE VECTOR THAT WILL CONTAIN THE FLOATING POINT
*         REPRESENTATION OF THE ASCII STRING CONTAINED IN BUFFER(2) THRU
*         BUFFER(8).
*
*****************************************************************************/
```

```
ASCII$TO$FLOAT:PROCEDURE(A,LIMIT,B) PUBLIC;
   DCL (A,B) ADDRESS,
       BUFFER BASED A BYTE,
       VECTOR BASED B (4) BYTE,
       TEN$FLOAT (4) BYTE DATA (00H,00H,20H,41H),
       TEMP (4) BYTE,
       RESULT (4) BYTE,
       (I,NUMINT,NUMDEC,NUM,LIMIT,T0,T1) BYTE;

   T0 = BUFFER;
   A = A + 1;
   T1 = BUFFER;
   DO I = 0 TO 3;              /* INITIALIZE VECTOR.  */
      VECTOR(I) = 0;
      END;
/*   CHECK IF PROPOSED NUMBER IS 0.     */
   IF T0 = 0 THEN RETURN;
   NUMINT = T1;
   NUMDEC = T0 - T1;
/*   FIND INTEGER PORTION OF NUMBER.    */
   DO WHILE NUMINT > 0;
      DO I = 0 TO 3;
         TEMP(I) = 0;
         END;
      A = A + 1;
      TEMP(0) = BUFFER - 30H;
      CALL FLTDS(.TEMP,.RESULT);
      IF NUMINT > 1 THEN DO;
         DO I = 1 TO NUMINT - 1;
            CALL FMUL(.RESULT,.TEN$FLOAT,.RESULT);
            END;
```

```
         END;
      CALL FADD(.RESULT,.VECTOR,.VECTOR);
      NUMINT = NUMINT - 1;
      END;

/*   FIND DECIMAL PORTION OF NUMBER PROPOSED.    */
   NUM = 0;
   DO WHILE NUMDEC > 0;
      DO I = 0 TO 3;
           TEMP(I) = 0;
           END;
      A = A + 1;
      NUM = NUM + 1;
      TEMP(0) = BUFFER - 30H;
      CALL FLTDS(.TEMP,.RESULT);
      DO I = 1 TO NUM;
         CALL FDIV(.RESULT,.TEN$FLOAT,.RESULT);
         END;
      CALL FADD(.RESULT,.VECTOR,.VECTOR);
      NUMDEC = NUMDEC - 1;
      END;

/*   CHECK FOR SIGN OF NUMBER PROPOSED.    */
   A = A + 1;
   IF BUFFER = 1 THEN VECTOR(3) = VECTOR(3) XOR 080H;

/*   ALL DONE.  RETURN TO CALLING MODULE.  */
   RETURN;
   END ASCII$TO$FLOAT;
```

```
/*************************************************************************************
*
* FRAC$TO$ASCII:
*   PROCEDURE USED TO CONVERT A FRACTIONAL PART OF A F.P.  NUMBER
*   INTO AN ASCII REPRESENTATION AND STORE IT IN A BUFFER.
*
* PARAMETERS:
*   - A -POINTER TO A VARIABLE - WHICH CONTAINS THE FRACTIONAL PART OF A
*         F.P.  NUMBER- PASSED BY FLOAT$TO$ASCII PROCEDURE.
*   - B -POINTER TO A BUFFER PARTIALLY FILLED W/ THE INTEGER PART OF
*        A F.P.  NUMBER CONVERTED TO ASCII BY FLOAT$TO$ASCII.
*   - DEC$POS -POINTER TO A VARIABLE WHICH INDICATES THE POSITION OF DECI-
*              MAL POINT TO BE SET ;  IT RETURN THE ADDRESS WHICH CONTAINS
*              THE FIRST VACANT POSITION IN THE BUFFER AFTER FILLED.
*
*************************************************************************************/
FRAC$TO$ASCII: PROCEDURE (A, B, DEC$POS);
   DCL (A, B, DEC$POS) ADDRESS;
   DCL TERM BASED A (4) BYTE,
       BUFF BASED B (28) BYTE,
       C BASED DEC$POS BYTE;
   DCL TEN$FLOAT (4) BYTE DATA (00H, 00H, 20H, 41H);
   DCL TEMP (4) BYTE, TEMP1 BYTE DATA (4),
       (FLAG, I) BYTE;

   BUFF(C) = '.';
   C = C + 1;
   FLAG = 0FFH;
   DO WHILE FLAG;            /* SET UP FRAC PART OF F.P.  NUMBER TO ASCII */
      CALL FMUL (.TERM, .TEN$FLOAT, .TERM);
      CALL FIXSD (.TERM, .TEMP);
```

```
        BUFF(C) = TEMP(0) + 30H;
        CALL FLTDS (.TEMP, .TEMP);
        CALL FSUB (.TERM, .TEMP, .TERM);
        C = C + 1;
        IF C = 26 THEN RETURN;
        FLAG = NOT (FLAG:= FZTST (.TERM, .TEMP1));
        END;
    RETURN;
    END FRAC$TO$ASCII;
```

```
/*****************************************************************************
*
* FLOAT$TO$ASCII:
*   PROCEDURE USED TO CONVERT A FLOATING POINT NUMBER -SEE INTEL SBC 310-
*    INTO AN ASCII REPRESENTATION AND STORE IT IN A BUFFER TO BE SENT TO THE
*   CALLING MODULE.
*
* PARAMETERS:
*   - FLOAT -POINTER TO A VARIABLE -WHICH CONTAINS THE F.P.  NUMBER- PASSED
*             BY THE CALLING MODULE;
*   - ASC$BUFFER -POINTER TO A BUFFER W/ LENGTH 28 WHICH CONTAINS THE ASCII
*                 REPRESENTATION OF THE F.P.  NO.;
*   - END$BUFFER - POINTER TO VARIABLE PASSED TO THE CALLING MODULE INDICA-
*                  TING THE FIRST VACANT BYTE OF BUFFER AFTER LAST PRINTABLE
*                  ASCII BYTE.
*
*****************************************************************************/
FLOAT$TO$ASCII: PROCEDURE (FLOAT, ASC$BUFFER, END$BUFFER) PUBLIC;
   DCL (FLOAT, ASC$BUFFER, END$BUFFER) ADDRESS,
            M BASED FLOAT (4) BYTE,
            A BASED ASC$BUFFER (28) BYTE,
            DEC$POINT BASED END$BUFFER BYTE,
            ONE (4) BYTE DATA (00H, 00H, 80H, 3FH),
            TEN (2) BYTE DATA (0AH, 00H),
            (TEMP, TEMP2, FRAC, REM) (4) BYTE,
            (TEMP1, I, SIGN, EXP, FLAG) BYTE;

   DO I = 0 TO LAST(A);                    /* INITIALIZE BUFFER */
        A(I) = ' ';
        END;
   SIGN = M(3) AND 80H;
```

```
    IF SIGN = 80H THEN A(0) = '-';
                  ELSE A(0) = ' ';
    EXP = SHL(M(3),1) OR SHR(M(2),7);
    TEMP(3) = M(3); TEMP(2) = M(2); TEMP(1) = M(1); TEMP(0) = M(0);
    IF ((EXP < 7FH) AND (EXP >= 00H)) THEN I = 0;
    IF EXP = 7FH THEN I = 1;
    IF ((EXP <= 0FEH) AND (EXP > 7FH)) THEN I = 2;
    DO CASE I;
      DO;                  /* CASE 0: APPLIED FOR -1.0 < F.P.  NUMBERS < 1.0 */
      DEC$POINT = 2;
      A(1) = '0';
      TEMP(3) = TEMP(3) AND 7FH;
      CALL FRAC$TO$ASCII (.TEMP,.A,.DEC$POINT);
      END;

      DO;
/* CASE 1: APPLIED FOR 2.0 > F.P.  NUMBER >= 1.0
           OR -2.0 < F.P.  NUMBER <= -1.0 */
      DEC$POINT = 2;
      A(1) = '1';
      TEMP(3) = TEMP(3) AND 7FH;
      CALL FSUB (.TEMP,.ONE,.TEMP);
      CALL FRAC$TO$ASCII (.TEMP,.A,.DEC$POINT);
      END;

      DO;
/* CASE 2: APPLIED FOR -1.0 > F.P.  NUMBER > 1.0 */
      DEC$POINT = 1;
        EXP = EXP - 7FH;           /* TAKE OUT BIAS OF EXPONENT */
        DO I = 1 TO 23 - EXP;
/* NEXT 2 INTERACTIVES "DO" TAKE OUT FRACTIONAL PART OF F.P.  NUMBER */
```

```
        TEMP(2) = SCR(TEMP(2),1);
        TEMP(1) = SCR(TEMP(1),1);
        IF CARRY THEN TEMP(0) = SHR(TEMP(0),1) OR 80H;
                 ELSE TEMP(0) = SHR(TEMP(0),1);
     END;
EXP = EXP OR EXP;           /* RESET CARRY BIT */
DO I = 1 TO 23 -EXP;
     TEMP(0) = SCL(TEMP(0),1);
     TEMP(1) = SCL(TEMP(1),1);
     IF CARRY THEN TEMP(2) = SHL(TEMP(2),1) OR 01H;
              ELSE TEMP(2) = SHL(TEMP(2),1);
     END;
CALL FSUB (.M,.TEMP,.FRAC);          /* SAVE FRAC PART OF F.P. NUMBER */
FRAC(3) = FRAC(3) AND 7FH;
CALL FIXSD (.TEMP,.TEMP);
IF (TEMP1 := TEMP(3) AND 80H) = 80H THEN
     DO;          /* PERFORMS TWO'S COMPLEMENT OF A NEGATIVE F.P. NUMBER */
     TEMP(0) = NOT TEMP(0); TEMP(1) = NOT TEMP(1);
     TEMP(2) = NOT TEMP(2); TEMP(3) = NOT TEMP(3);
     TEMP(0) = TEMP(0) + 1; TEMP(1) = TEMP(1) PLUS 0;
     TEMP(2) = TEMP(2) PLUS 0; TEMP(3) = TEMP(3) PLUS 0;
     END;
FLAG = 0FFH;
DO WHILE FLAG;          /* SET UP INTEGER PART OF F.P. NUMBER TO ASCII */
   CALL EDIV (.TEMP,.TEN,.TEMP,.REM);
   A(DEC$POINT) = REM(0) + 30H;
   DEC$POINT = DEC$POINT + 1;
   CALL FLTDS (.TEMP,.TEMP);
   TEMP1 = 4;
   FLAG = NOT (FLAG:= FZTST (.TEMP, .TEMP1));
   CALL FIXSD (.TEMP,.TEMP);
```

```
            END;
         DO I = 1 TO DEC$POINT/2;
            TEMP1 = A(I);
            A(I) = A(DEC$POINT - I);
            A(DEC$POINT - I) = TEMP1;
            END;
         CALL FRAC$TO$ASCII (.FRAC,.A,.DEC$POINT);
         END;

      END;
   RETURN;
   END FLOAT$TO$ASCII;

END FLTASCII;
```

FLOATING$POINT

```
FLOATING$POINT: DO;
```

```
/***   DECLARATIONS:   ***/

DECLARE LIT LITERALLY 'LITERALLY',
        DCL LIT 'DECLARE';
DCL
    MEBAS ADDRESS PUBLIC DATA (0F790H) ,
    RES$TABLE (8) BYTE PUBLIC AT (0F790H), /* MEMORY RESERVED FOR F.P. BOARD. */
    OUT$OP$CODE LIT '010H',   /* BASE OUTPUT PORT FOR F.P. BOARD. */
    IN$STATUS LIT '011H',     /* INPUT PORT FOR STATUS OF F.P. BOARD. */
    IN$FLAG LIT '017H',       /* INPUT PORT FOR FLAG FROM F.P. BOARD. */
    MEM$LOW LIT '011H',       /* OUTPUT PORT FOR LOW PORTION OF MEMORY BASE. */
    MEM$HIGH LIT '012H',      /* OUTPUT PORT FOR HIGH PORTION OF MEMORY BASE. */
    MUL$CODE LIT '00H',       /* FIXED POINT MULTIPLICATION CODE. */
    DIV$CODE LIT '01H',       /* FIXED POINT DIVISION CODE. */
    FMUL$CODE LIT '02H',      /* F.P. MULTIPLICATION CODE. */
    FDIV$CODE LIT '03H',      /* F.P. DIVISION CODE. */
    FADD$CODE LIT '04H',      /* F.P. ADD CODE. */
    FSUB$CODE LIT '05H',      /* F.P. SUBTRACT CODE. */
    FSQR$CODE LIT '06H',      /* F.P. SQUARE CODE. */
    FSQRT$CODE LIT '07H',     /* F.P. SQUARE ROOT CODE. */
    FLTDS$CODE LIT '08H',     /* FIXED-TO-FLOAT CONVERSION CODE. */
    FIXSD$CODE LIT '09H',     /* FLOAT-TO-FIXED CONVERSION CODE. */
    FCMPR$CODE LIT '0AH',     /* F.P. COMPARE CODE. */
    FZTST$CODE LIT '0BH',     /* F.P. TEST CODE. */
    EXCH$CODE LIT '0FH',      /* EXCHANGE CODE. */
    EDIV$CODE LIT '0EH',      /* EXTENDED FIXED POINT DIVISION CODE. */
    BUSY$MASK LIT '01H',      /* MASK FOR BUSY SIGNAL FROM F.P. BOARD. */
    ERROR$MASK LIT '04H';     /* MASK FOR ERROR CODE FROM F.P. BOARD. */
```

```
/***    EXTERNALS:    ***/

CRT$PRINT$STRING:PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS;
   END CRT$PRINT$STRING;

SEND$CRLF: PROCEDURE EXTERNAL;
   END SEND$CRLF;
```

```
/*****************************************************************************
*
* INIT$FP:
*   PROCEDURE USED TO INITIALIZE THE FLOATING POINT MODULE.
*
* USAGE:
*   THIS PROCEDURE SHOULD BE CALLED ONE TIME FROM THE USER'S
*   MAIN PROGRAM BEFORE ATTEMPTING TO USE ANY OF THE ROUTINES PROVI-
*   DED FOR FLOATING POINT OPERATIONS WITH THE FLOATING POINT BOARD.
*
*****************************************************************************/
INIT$FP: PROCEDURE PUBLIC;
   OUTPUT(MEM$LOW) = LOW(MEBAS);
   OUTPUT(MEM$HIGH) = HIGH(MEBAS);
   RETURN;
   END INIT$FP;
```

```
/*****************************************************************************
*
* ADJUST$OP:
*   PROCEDURE USED TO PUT TWO VECTORS INTO THE TABLE VECTOR.
*
* PARAMETERS:
*   - A,B. - POINTERS TO FIRST AND SECOND VECTOR VALUES.
*
*****************************************************************************/
ADJUST$OP: PROCEDURE(A,B);
   DCL (A,B) ADDRESS,
       OP1 BASED A (4) BYTE,
       OP2 BASED B (4) BYTE,
       I BYTE;
   DO I = 0 TO LAST(OP1);
      RES$TABLE(I) = OP1(I);
      RES$TABLE(I + 4) = OP2(I);
      END;
   RETURN;
   END ADJUST$OP;
```

```
/*****************************************************************************
*
* ADJUST1$OP:
*   PROCEDURE USED TO PUT ONE VECTOR INTO THE TABLE VECTOR.
*
* PARAMETERS:
*   -  A. - POINTER TO VECTOR VALUE.
*
*****************************************************************************/
ADJUST1$OP: PROCEDURE (A);
   DCL A ADDRESS,
       OP1 BASED A (4) BYTE,
       I BYTE;
   DO I = 0 TO LAST(OP1);
      RES$TABLE(I) = OP1(I);
      END;
   RETURN;
   END ADJUST1$OP;
```

```
/*****************************************************************************
*
* ADJUST2$OP:
*   PROCEDURE USED TO PUT TWO ADDRESS VALUES INTO THE TABLE VECTOR.
*
* PARAMETERS:
*   - A,B. - POINTERS TO TWO ADDRESS VALUES.
*
*****************************************************************************/
ADJUST2$OP: PROCEDURE (A,B);
   DCL (A,B) ADDRESS,
       OP1 BASED A (2) BYTE,
       OP2 BASED B (2) BYTE,
       I BYTE;
   DO I = 0 TO LAST(OP1);
      RES$TABLE(I) = OP1(I);
      RES$TABLE(I + 4) = OP2(I);
      END;
   RETURN;
   END ADJUST2$OP;
```

```
/*****************************************************************
*
* VAL$RESULT:
*   PROCEDURE USED TO GET THE RESULT IN FIRST FOUR BYTES OF THE
*   TABLE VECTOR, AND PUT IT INTO ANOTHER VECTOR PROVIDED.
*
* PARAMETERS:
*   - C. - POINTER TO VECTOR ON WHICH RESULT IS DESIRED.
*
*****************************************************************/
VAL$RESULT: PROCEDURE(C);
   DCL C ADDRESS,
       RESULT BASED C (4) BYTE,
       I BYTE;
   DO I = 0 TO LAST(RESULT);
      RESULT(I) = RES$TABLE(I);
      END;
   RETURN;
   END VAL$RESULT;
```

```
/*****************************************************************************
*
* VAL$RESULT$1:
*   PROCEDURE USED TO PUT ALL EIGHT BYTES OF THE TABLE VECTOR
*   INTO TWO FOUR BYTES VECTORS PROVIDED.
*
* PARAMETERS:
*   - A,B. - POINTERS TO VECTORS ON WHICH RESULT IS DESIRED.  A
*     MUST POINT TO FIRST VECTOR (FIRST FOUR BYTES OF
*     TABLE), AND B MUST POINT TO THE SECOND VECTOR (LAST FOUR
*     BYTES IN TABLE).
*
*****************************************************************************/
VAL$RESULT$1: PROCEDURE(A,B);
   DCL (A,B) ADDRESS,
       OP1 BASED B (4) BYTE,
       OP2 BASED A (4) BYTE,
       I BYTE;
   DO I = 0 TO LAST(OP1);
      OP2(I) = RES$TABLE(I);
      OP1(I) = RES$TABLE(I + 4);
      END;
   RETURN;
   END VAL$RESULT$1;
```

```
/*****************************************************************************
*
* VAL$RESULT$2:
*   PROCEDURE USED TO GET THE RESULT FROM FIXED POINT
*   DIVISION OPERATION, AND PUT THEM INTO TWO ADDRESS
*   LOCATIONS PROVIDED.
*
* PARAMETERS:
*   - C,R. - POINTERS TO TWO ADDRESS LOCATIONS IN WHICH THE
*     RESULT IS DESIRED TO BE PLACED.
*
*****************************************************************************/
VAL$RESULT$2: PROCEDURE (C,R);
   DCL (C,R) ADDRESS,
       OP1 BASED C (2) BYTE,
       OP2 BASED R (2) BYTE,
       I BYTE;
   DO I = 0 TO LAST(OP1);
      OP1(I) = RES$TABLE(I);
      OP2(I) = RES$TABLE(I + 4);
      END;
   END VAL$RESULT$2;
```

```
/*****************************************************************************
*
* COMPARE:
*   PROCEDURE USED TO CHECK FOR OUTPUT CONDITIONS FROM F.P.  BOARD.
*
* PARAMETERS:
*   - A. - BYTE VALUE CONTAINING RESULT FROM F.P.  BOARD.
*   - B. - BYTE VALUE CONTAINING CONDITION DESIRED TO BE CHECKED:
*             <  ..... 0
*             <= ..... 1
*             >  ..... 2
*             >= ..... 3
*             =  ..... 4
*             <> ..... 5
* USAGE:
*   TYPED PROCEDURE THAT RETURNS A 'TRUE' VALUE (001H) IF OUTPUT
*   CONDITION FROM F.P.  BOARD AND CONDITION DESIRED TO BE TESTED
*   ARE SIMILAR.  IF NOT SIMILAR, A 'FALSE' VALUE (000H) IS RETURNED.
*
*****************************************************************************/
COMPARE: PROCEDURE (A,B) BYTE;
   DCL TRUE LIT '01H',
       FALSE LIT '00H';
   DCL (A,B) BYTE,
       (LESS$THAN,LESS$OR$EQUAL,GREATER$THAN,GREAT$OR$EQUAL,EQUAL,NOT$EQUAL)
       BYTE DATA (0,1,2,3,4,5);
   IF ((A = 80H) AND ((B = LESS$OR$EQUAL) OR (B = GREAT$OR$EQUAL) OR
      (B = EQUAL))) THEN RETURN TRUE;
   IF ((A = 40H) AND ((B = GREATER$THAN) OR (B = GREAT$OR$EQUAL) OR
      (B = NOT$EQUAL))) THEN RETURN TRUE;
   IF ((A = 20H) AND ((B = LESS$THAN) OR (B = LESS$OR$EQUAL) OR
```

```
    (B = NOT$EQUAL)) THEN RETURN TRUE;
RETURN FALSE;
END COMPARE;
```

```
/*****************************************************************************
*
* FLOAT$MSG$ERROR:
*   PROCEDURE USED TO SEND A MESSAGE ERROR ACCORDING TO ERROR
*   CODE PROVIDED BY F.P.  BOARD.
*
* USAGE:
*   UNTYPED PROCEDURE.  IF CONDITION OF ERROR IS DETECTED, THIS
*   PROCEDURE MUST BE CALLED IN ORDER TO OBTAIN ERROR CODE AND
*   DISPLAY AN APPROPRIATE MESSAGE.  NOTE THAT PROGRAM EXECUTION
*   WILL BE STOPPED AND INTERRUPTS WILL BE ENABLED IF AN ERROR
*   IS DETECTED.
*
*****************************************************************************/
FLOAT$MSG$ERROR: PROCEDURE;
   DCL MSG1(*) BYTE DATA ('DIVISION BY ZERO.$$'),
       MSG2(*) BYTE DATA ('DOMAIN ERROR.$$'),
       MSG3(*) BYTE DATA ('OVERFLOW.$$'),
       MSG4(*) BYTE DATA ('UNDERFLOW.$$'),
       MSG5(*) BYTE DATA ('INVALID FORMAT FOR FIRST ARGUMENT.$$'),
       MSG6(*) BYTE DATA ('INVALID FORMAT FOR SECOND ARGUMENT.$$'),
       ERROR(*) BYTE DATA (' FATAL ERROR.$$'),
       I BYTE;
   I = INPUT(IN$STATUS) AND 07H;
   DO CASE I;
      ;
      CALL CRT$PRINT$STRING(.MSG1);
      CALL CRT$PRINT$STRING(.MSG2);
      CALL CRT$PRINT$STRING(.MSG3);
      CALL CRT$PRINT$STRING(.MSG4);
      CALL CRT$PRINT$STRING(.MSG5);
```

```
        CALL CRT$PRINT$STRING(.MSG6);
        ;
        END;
    CALL CRT$PRINT$STRING(.ERROR);
    CALL SEND$CRLF;
    RETURN;
    END FLOAT$MSG$ERROR;
```

```
/*****************************************************************************
*
*
* CHECK:
*   PROCEDURE USED TO CHECK FOR STATUS OF F.P.  BOARD AND TO DETECT IF
*   ANY ERROR HAS OCCURRED.
*
* USAGE:
*   UNTYPED PROCEDURE THAT IS CALLED BY ALL PROCEDURES THAT TRY TO EXECUTE
*   A FLOATING POINT OPERATION WITH THE F.P.  BOARD.
*
*****************************************************************************/
CHECK: PROCEDURE;
   DCL I BYTE;
   DO WHILE ((I:=INPUT(IN$FLAG)) AND BUSY$MASK) = BUSY$MASK;
      END;
   IF (I AND ERROR$MASK) = ERROR$MASK
      THEN DO;
         CALL FLOAT$MSG$ERROR;
         HALT;
         END;
   RETURN;
   END CHECK;
```

```
/*****************************************************************************
*
* MUL:
*   PROCEDURE USED TO PERFORM FIXED POINT MULTIPLICATION USING THE F.P.
*   BOARD.
*
* PARAMETERS:
*   - A. - POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE FIRST
*          OPERAND WILL BE LOCATED.
*   - B. - POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE SECOND
*          OPERAND WILL BE LOCATED.
*   - C. - POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE RESULT
*          IS DESIRED TO BE PLACED.  NOTICE THAT IT COULD BE THE SAME
*          LOCATION USED FOR ANY OF THE OPERANDS.
*
*****************************************************************************/
MUL: PROCEDURE (A,B,C) PUBLIC;
   DCL (A,B,C) ADDRESS;
   CALL ADJUST2$OP (A,B);
   OUTPUT(OUT$OP$CODE) = MUL$CODE;
   CALL CHECK;
   CALL VAL$RESULT (C);
   RETURN;
   END MUL;
```

```
/*****************************************************************************
*
* DIV:
*   PROCEDURE USED TO PERFORM FIXED POINT DIVISION USING THE
*   F.P. BOARD.
*
* PARAMETERS:
*   - A. - POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE DIVI-
*     DEND IS LOCATED.
*   - B. - POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE DIVI-
*     SOR IS LOCATED.
*   - C. - POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE RESULT
*     (QUOTIENT) IS DESIRED TO BE PLACED.
*   - R. - POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE REMAIN-
*     DER IS DESIRED TO BE PLACED.  NOTICE THAT C AND R COULD POINT TO ANY
*     OF THE TWO OPERANDS IF SO DESIRED.
*
*****************************************************************************/
DIV: PROCEDURE (A,B,C,R) PUBLIC;
    DCL (A,B,C,R) ADDRESS;
    CALL ADJUST2$OP(A,B);
    OUTPUT(OUT$OP$CODE) = DIV$CODE;
    CALL CHECK;
    CALL VAL$RESULT$2(C,R);
    RETURN;
    END DIV;
```

```
/*****************************************************************************
*
* EDIV:
*   PROCEDURE USED TO PERFORM EXTENDED FIXED POINT DIVISION USING THE
*   F.P.  BOARD.
*
* PARAMETERS:
*   - A. - POINTER TO A 4 BYTE VECTOR THAT WILL PROVIDE THE DIVIDEND.
*   - B. - POINTER TO A 2 BYTE VECTOR THAT WILL PROVIDE THE DIVISOR.
*   - C. - POINTER TO A 4 BYTE VECTOR IN WHICH THE QUOTIENT WILL BE RETURNED.
*   - R. - POINTER TO A 4 BYTE VECTOR IN WHICH THE REMAINDER WILL BE RETURNED.
*
*****************************************************************************/
EDIV: PROCEDURE (A,B,C,R) PUBLIC;
   DCL (A,B,C,R) ADDRESS,
       OP2 BASED B (2) BYTE,
       I BYTE;
   CALL ADJUST1$OP (A);
   DO I = 0 TO LAST(OP2);
      RES$TABLE(I + 4) = OP2(I);
      END;
     OUTPUT(OUT$OP$CODE) = EDIV$CODE;
   CALL CHECK;
   CALL VAL$RESULT$1 (C,R);
   RETURN;
   END EDIV;
```

```
/*****************************************************************************
*
* FMUL:
*   PROCEDURE USED TO PERFORM FLOATING POINT MULTIPLICATION USING THE
*   F.P.  BOARD.
*
* PARAMETERS:
*   -A,B,C. - POINTERS TO 3 FOUR BYTE VECTORS THAT POINT TO THE TWO OPERANDS
*    AND THE RESULT RESPECTIVELY.  NOTE THAT THE RESULT COULD BE THE SAME
*    VECTOR USED TO INDICATE ANY OPERAND.
*
*****************************************************************************/
FMUL: PROCEDURE (A,B,C) PUBLIC;
   DCL (A,B,C) ADDRESS;
   CALL ADJUST$OP(A,B);
   OUTPUT(OUT$OP$CODE) = FMUL$CODE;
   CALL CHECK;
   CALL VAL$RESULT(C);
   RETURN;
   END FMUL;
```

```
/*****************************************************************************
*
* FDIV:
*   PROCEDURE USED TO PERFORM FLOATING POINT DIVISION USING THE F.P BOARD.
*
* PARAMETERS:
*   - A,B,C. - POINTERS TO 3 FOUR BYTE VECTORS THAT WILL POINT TO THE DIVIDEND
*     AND THE DIVISOR ON THE FIRST TWO, AND THE RESULT ON THE THIRD.  NOTE
*     THAT THE RESULT COULD BE PUT IN THE SAME PLACE OF ANY OPERAND IF SO
*     DESIRED.
*
*****************************************************************************/
FDIV: PROCEDURE (A,B,C) PUBLIC;
   DCL (A,B,C) ADDRESS;
   CALL ADJUST$OP (A,B);
   OUTPUT(OUT$OP$CODE) = FDIV$CODE;
   CALL CHECK;
   CALL VAL$RESULT(C);
   RETURN;
   END FDIV;
```

```
/*****************************************************************************
*
* FADD:
*   PROCEDURE USED TO PERFORM FLOATING POINT ADDITION USING THE F.P.  BOARD.
*
* PARAMETERS:
*   - A,B,C. - POINTERS TO 3 FOUR BYTE VECTORS.  THE FIRST TWO POINT TO THE
*     TWO OPERANDS, AND THE THIRD ONE POINTS TO THE VECTOR RESULT.  NOTE
*     THAT THE RESULT COULD ALSO BE PLACED IN ANY OF THE OPERAND VECTORS.
*
*****************************************************************************/
FADD: PROCEDURE (A,B,C) PUBLIC;
   DCL (A,B,C) ADDRESS;
   CALL ADJUST$OP (A,B);
   OUTPUT(OUT$OP$CODE) = FADD$CODE;
   CALL CHECK;
   CALL VAL$RESULT (C);
   RETURN;
   END FADD;
```

```
/*****************************************************************************
*
* FSUB:
*   PROCEDURE USED TO PERFORM FLOATING POINT SUBTRACTION USING THE
*   F.P.  BOARD.
*
* PARAMETERS:
*   - A,B,C. - POINTERS TO 3 FOUR BYTE VECTORS.  THE FIRST TWO POINT TO
*     BOTH OPERANDS, AND THE THIRD ONE POINTS TO THE VECTOR WHERE THE
*     RESULT IS DESIRED TO BE PLACED.  NOTE THAT THE RESULT COULD BE
*     PLACED IN ANY OF BOTH OPERANDS.
*
*****************************************************************************/
FSUB: PROCEDURE (A,B,C) PUBLIC;
   DCL (A,B,C) ADDRESS;
   CALL ADJUST$OP (A,B);
   OUTPUT(OUT$OP$CODE) = FSUB$CODE;
   CALL CHECK;
   CALL VAL$RESULT (C);
   RETURN;
   END FSUB;
```

```
/*****************************************************************************
*
* FSQR:
*   PROCEDURE USED TO PERFORM A FLOATING POINT SQUARE USING THE F.P.
*   BOARD.
*
* PARAMETERS:
*   - A,C. - POINTERS TO 2 FOUR BYTE VECTORS.  A POINTS TO A VECTOR IN WHICH
*     A FLOATING POINT QUANTITY IS LOCATED AND TO WHICH ITS SQUARE WILL BE
*     OBTAINED.  C POINTS TO A VECTOR IN WHICH THE RESULT IS DESIRED TO BE
*     PLACED.  NOTE THAT A AND C COULD POINT TO THE SAME VECTOR.
*
*****************************************************************************/
FSQR: PROCEDURE (A,C)  PUBLIC;
   DCL (A,C) ADDRESS;
   CALL ADJUST1$OP(A);
   OUTPUT(OUT$OP$CODE) = FSQR$CODE;
   CALL CHECK;
   CALL VAL$RESULT (C);
   RETURN;
   END FSQR;
```

```
/*****************************************************************************
*
* FLTDS:
*   PROCEDURE USED TO PERFORM A FIXED-TO-FLOAT CONVERSION USING THE F.P.
*   BOARD.
*
* PARAMETERS:
*   - A,C. - POINTERS TO 2 FOUR BYTE VECTORS.  A POINTS TO A VECTOR CONTAINING
*     A FIXED POINT INTEGER AND C POINTS TO A VECTOR WHERE THE SINGLE PRE-
*     CISION FLOATING POINT REPRESENTATION OF THE SAME VALUE, IS DESIRED TO
*     BE PLACED.  NOTE THAT BOTH, A AND C, COULD POINT TO THE SAME VECTOR.
*
*****************************************************************************/
FLTDS: PROCEDURE (A,C) PUBLIC;
   DCL (A,C) ADDRESS;
   CALL ADJUST1$OP (A);
   OUTPUT(OUT$OP$CODE) = FLTDS$CODE;
   CALL CHECK;
   CALL VAL$RESULT (C);
   RETURN;
   END FLTDS;
```

```
/*****************************************************************************
*
* FIXSD:
*   PROCEDURE USED TO PERFORM A FLOAT-TO-FIXED CONVERSION USING THE F.P.
*   BOARD.
*
* PARAMETERS:
*   - A,C. - POINTERS TO 2 FOUR BYTE VECTORS.  A POINTS TO A VECTOR CONTAINING
*     THE FIXED POINT QUANTITY DESIRED TO BE CONVERTED.  C POINTS TO A VECTOR
*     IN WHICH THE CONVERSION IS DESIRED TO BE PLACED.  NOTE THAT BOTH COULD
*     POINT TO THE SAME VECTOR.
*
*****************************************************************************/
FIXSD: PROCEDURE (A,C) PUBLIC;
   DCL (A,C) ADDRESS;
   CALL ADJUST1$OP (A);
   OUTPUT(OUT$OP$CODE) = FIXSD$CODE;
   CALL CHECK;
   CALL VAL$RESULT (C);
   RETURN;
   END FIXSD;
```

```
/*****************************************************************************
*
* FSQRT:
*   PROCEDURE TO PERFORM THE SQUARE ROOT OF A FLOATING POINT NUMBER USING
*   THE F.P.  BOARD.
*
* PARAMETERS:
*   - A,C. - POINTERS TO 2 FOUR BYTE VECTORS.  A POINTS TO THE VECTOR CONTAINING
*     THE NUMBER TO WHICH ITS SQUARE ROOT WILL BE OBTAINED.  C POINTS TO THE
*     VECTOR IN WHICH THE RESULT WILL BE PLACED.  NOTE THAT BOTH POINTERS
*     COULD BE REFERING TO THE SAME VECTOR.
*
*****************************************************************************/
FSQRT: PROCEDURE (A,C)  PUBLIC;
   DCL (A,C) ADDRESS;
   CALL ADJUST1$OP (A);
   OUTPUT(OUT$OP$CODE) = FSQRT$CODE;
   CALL CHECK;
   CALL VAL$RESULT (C);
   RETURN;
   END FSQRT;
```

```
/*****************************************************************************
*
* FCMPR:
*   PROCEDURE USED TO COMPARE TWO FLOATING POINT NUMBERS USING THE F.P.  BOARD.
*
* PARAMETERS:
*   - A,B. - POINTERS TO 2 FOUR BYTE VECTORS CONTAINING THE TWO FLOATING
*     POINT NUMBERS DESIRED TO BE COMPARED.
*   - C. - POINTS TO A BYTE VALUE CONTAINING THE CODE CORRESPONDING TO THE
*     TYPE OF COMPARISON DESIRED:
*                    <  ..... 0
*                    <= ..... 1
*                    >  ..... 2
*                    >= ..... 3
*                    =  ..... 4
*                    <> ..... 5
*
* USAGE:
*   TYPED PROCEDURE.  IF THE RELATION DESIRED TO BE TESTED HOLDS, A VALUE OF
*   '01H' (TRUE) IS RETURNED, OTHERWISE A VALUE OF '00H' (FALSE) WILL BE
*   RETURNED.
*
*****************************************************************************/
FCMPR: PROCEDURE (A,B,C) BYTE PUBLIC;
   DCL (A,B,C) ADDRESS,
       (TEST BASED C,RESULT, FLAG) BYTE;
   FLAG = 000H;       /* RESET FLAG USED TO CHECK TWO NEGATIVE NUMBERS.  */
   CALL ADJUST$OP (A,B);
   IF (RES$TABLE(3) >= 80H) AND
      (RES$TABLE(7) >= 80H)
      THEN FLAG = 0FFH;
```

```
OUTPUT(OUT$OP$CODE) = FCMPR$CODE;
CALL CHECK;
RESULT = INPUT (IN$STATUS) AND 0E0H;
IF FLAG AND (RESULT <> 80H)
THEN DO;
     IF RESULT = 40H
        THEN RESULT = 20H;
        ELSE RESULT = 40H;
     END;
 RETURN (RESULT:= COMPARE(RESULT,TEST));
END FCMPR;
```

```
/*****************************************************************************
*
* FZTST:
*   PROCEDURE USED TO TEST A FLOATING POINT NUMBER AGAINST 0.0 USING THE F.P.
*   BOARD.
*
* PARAMETERS:
*   - A. - POINTER TO A FOUR BYTE VECTOR CONTAINING THE FLOATING POINT VALUE
*     DESIRED TO BE TESTED.
*   - C. - POINTER TO A BYTE VALUE CONTAINING THE CODE OF THE COMPARISON DESI-
*     RED, ACCORDING TO THE FOLLOWING RULES:
*                    <  ..... 0
*                    <= ..... 1
*                    >  ..... 2
*                    >= ..... 3
*                    =  ..... 4
*                    <> ..... 5
*
* USAGE:
*   TYPED PROCEDURE.  IF THE RELATION DESIRED TO BE TESTED HOLDS, A VALUE OF
*   '01H' (TRUE) IS RETURNED, OTHERWISE A VALUE OF '00H' (FALSE) WILL BE RE-
*   TURNED.
*
*****************************************************************************/
FZTST: PROCEDURE (A,C) BYTE PUBLIC;
   DCL (A,C) ADDRESS,
       OP1 BASED A (4) BYTE,
       (TEST BASED C, RESULT, I) BYTE;
   DCL LOWER$BOUND (4) BYTE DATA (0E2H,0FAH,0FFH,084H), /* -0.0000009 */
       UPPER$BOUND (4) BYTE DATA (0E2H,0FAH,0FFH,034H), /* 0.0000009 */
       LOW BYTE DATA (01H),    /* LESS THAN OR EQUAL */
```

```
      HIGH BYTE DATA (03H);  /* GREATER THAN OR EQUAL */

   /* CHECK IF NUMBER IS IN BOUNDARIES OF DEFINED SYSTEM ZERO */
IF (TEST <> 0) AND (TEST <> 2) AND
   (FCMPR(.OP1, .LOWER$BOUND, .HIGH)) AND
   (FCMPR(.OP1, .UPPER$BOUND, .LOW))
THEN DO;
   DO I = 0 TO 3;
      OP1(I) = 00H;
      END;
   END;
CALL ADJUST1$OP (A);
OUTPUT(OUT$OP$CODE) = FZTST$CODE;
CALL CHECK;
RESULT = INPUT (IN$STATUS) AND 0E0H;
RETURN (RESULT:= COMPARE(RESULT,TEST));
END FZTST;
```

```
/*****************************************************************************
*
* EXCH:
*   PROCEDURE USED TO EXCHANGE TWO FLOATING POINT VALUES USING THE F.P.
*   BOARD.
*
* PARAMETERS:
*   - A,C. - POINTERS TO 2 FOUR BYTE VECTORS CONTAINING TWO FLOATING
*     POINT NUMBERS DESIRED TO BE EXCHANGED.
*
*****************************************************************************/
EXCH: PROCEDURE (A,C) PUBLIC;
   DCL (A,C) ADDRESS;
   CALL ADJUST$OP (A,C);
   OUTPUT(OUT$OP$CODE) = EXCH$CODE;
   CALL CHECK;
   CALL VAL$RESULT$1(A,C);
   RETURN;
   END EXCH;
```

```
/*****************************************************************************
*
* COS$SIN:
*   THIS PROCEDURE IS USED TO CALCULATE THE COSINE AND SINE FUNCTIONS
*   OF A GIVEN ARGUMENT IN RADIANS.
*
* PARAMETERS:
*   - A. - POINTER TO A FOUR BYTE VECTOR IN WHICH THE FLOATING POINT REPRE-
*     SENTATION OF AN ANGLE IN RADIANS IS LOCATED.
*
*   - C. - POINTER TO A FOUR BYTE VECTOR IN WHICH THE VALUE OF THE COSINE
*     OF THE GIVEN ANGLE, WILL BE PLACED.
*
*   - S. - POINTER TO A FOUR BYTE VECTOR IN WHICH THE VALUE OF THE SINE OF
*     THE GIVEN ANGLE WILL BE PLACED.
*
*****************************************************************************/
COS$SIN: PROCEDURE (A,C,S) PUBLIC;
  DCL (A,C,S) ADDRESS,
      ANGLE BASED A (4) BYTE,
      COSINE BASED C (4) BYTE,
      SINE BASED S (4) BYTE,
      ANGLE$SQ (4) BYTE,
      TEMP (4) BYTE,
      TEMP0 (4) BYTE,
      TEMP1 (4) BYTE,
      MINUS$ONE (4) BYTE DATA (00H,00H,80H,0BFH),    /* -1.0 */
      ONE$FLOAT (4) BYTE DATA (00H,00H,80H,3FH),     /* 1.0 IN F.P.  FORMAT. */
      TWO$FLOAT (4) BYTE DATA (00H,00H,00H,40H),     /* 2.0 */
      PI$FLOAT (4) BYTE DATA (0DBH,0FH,49H,40H),     /* 3.141593 */
      TWO$PI (4) BYTE DATA (0DBH,0FH,0C9H,40H),      /* 6.2831853 */
```

```
        PI$OVER2 (4) BYTE DATA (0DBH,0FH,0C9H,3FH),    /* 1.5707963 */
        PI$3$OVER2 (4) BYTE DATA (0E4H,0CBH,96H,40H), /* 4.7123889 */
        CONST$1 (4) BYTE DATA (0B5H,1FH,12H,3FH),      /* 0.5707963 */
        CONST$2 (4) BYTE DATA (0E6H,5DH,25H,0BFH),     /* -0.645964 */
        CONST$3 (4) BYTE DATA (0E1H,35H,0A3H,3DH),     /* 0.079629261 */
        CONST$4 (4) BYTE DATA (0AAH,68H,99H,0BBH),     /* -0.0046816668 */
        CONST$5 (4) BYTE DATA (25H,0BH,28H,39H),       /* 0.0001602588 */
        CONST$6 (4) BYTE DATA (0A4H,067H,66H,0B6H),    /* -0.0000034333 */
        CHECK BYTE DATA (002H),     /* CHECK FOR GREATER THAN */
        CHECK1 BYTE DATA (003H),    /* CHECK FOR GREATER THAN OR EQUAL */
        CHECK2 BYTE DATA (001H),    /* CHECK FOR LESS THAN OR EQUAL */
        CHECK3 BYTE DATA (004H),    /* CHECK FOR EQUAL */
        (SIGN,SIGN1,QUAD,I) BYTE;

    DO I = 0 TO 3;
        TEMP0(I) = ANGLE(I);
        SINE(I), COSINE(I) = 00H;
        END;
/*    CHECK IF ANGLE IS >= 360 DEGREES.     */
    SIGN = FCMPR(.TEMP0,.TWO$PI,.CHECK1);
    DO WHILE SIGN;
        CALL FSUB(.TEMP0,.TWO$PI,.TEMP0);
        SIGN = FCMPR(.TEMP0,.TWO$PI,.CHECK1);
        END;
/*    CHECK IF ANGLE IS NEGATIVE.     */
    DO WHILE TEMP0(3) >= 080H;
        CALL FADD(.TEMP0,.TWO$PI,.TEMP0);
        END;
/*    CHECK FOR SPECIAL CASES    */
    IF FCMPR (.TEMP0, .PI$OVER2, .CHECK3)
    THEN DO;      /* 90 DEGREES */
```

```
      DO I = 0 TO 3;
         SINE(I) = ONE$FLOAT(I);
         END;
      RETURN;
      END;
   IF FCMPR (.TEMP0, .PI$3$OVER2, .CHECK3)
   THEN DO;     /* 270 DEGREES */
      DO I = 0 TO 3;
         SINE(I) = MINUS$ONE(I);
         END;
      RETURN;
      END;
   IF FCMPR (.TEMP0, .TWO$PI, .CHECK3)
   THEN DO;     /* 360 DEGREES */
      DO I = 0 TO 3;
         COSINE(I) = ONE$FLOAT(I);
         END;
      RETURN;
      END;
/*    TO NORMALIZE THE ANGLE BETWEEN 0 AND 90 DEGREES.     */
   QUAD = 1;
   IF (SIGN := FCMPR(.TEMP0,.PI$OVER2,.CHECK)) AND
      (SIGN1 := FCMPR(.TEMP0,.PI$FLOAT,.CHECK2))
   THEN DO;
      QUAD = 2;
      CALL FSUB(.PI$FLOAT,.TEMP0,.TEMP0);
      END;
   ELSE DO;
      IF (SIGN := FCMPR(.TEMP0,.PI$FLOAT,.CHECK)) AND
         (SIGN1 := FCMPR(.TEMP0,.PI$3$OVER2,.CHECK2))
      THEN DO;
```

```
            QUAD = 3;
            CALL FSUB(.TEMP0,.PI$FLOAT,.TEMP0);
            END;
         ELSE DO;
            IF (SIGN := FCMPR(.TEMP0,.PI$3$OVER2,.CHECK))
            THEN DO;
               QUAD = 4;
               CALL FSUB(.TWO$PI,.TEMP0,.TEMP0);
               END;
            END;
         END;
/*    CONVERT ANGLE IN RADIANS TO ANGLE IN SEMICIRCLE UNITS.    */
   CALL FDIV(.TEMP0,.PI$FLOAT,.TEMP);
/*    GET THE SQUARE OF THE ANGLE.    */
   CALL FSQR(.TEMP,.ANGLE$SQ);
/*    PERFORM HASTINGS APPROXIMATION.    */
   CALL FMUL(.ANGLE$SQ,.CONST$6,.TEMP1);
   CALL FADD(.TEMP1,.CONST$5,.TEMP1);
   CALL FMUL(.ANGLE$SQ,.TEMP1,.TEMP1);
   CALL FADD(.TEMP1,.CONST$4,.TEMP1);
   CALL FMUL(.ANGLE$SQ,.TEMP1,.TEMP1);
   CALL FADD(.TEMP1,.CONST$3,.TEMP1);
   CALL FMUL(.ANGLE$SQ,.TEMP1,.TEMP1);
   CALL FADD(.TEMP1,.CONST$2,.TEMP1);
   CALL FMUL(.ANGLE$SQ,.TEMP1,.TEMP1);
   CALL FADD(.TEMP1,.CONST$1,.TEMP1);
   CALL FMUL(.TEMP,.TEMP1,.TEMP1);
   CALL FADD(.TEMP1,.TEMP,.TEMP1);
   CALL FSQR(.TEMP1,.TEMP1);
   CALL FMUL(.TEMP1,.TWO$FLOAT,.TEMP1);
/*    TO COMPUTE THE VALUE OF THE COSINE.    */
```

```
   CALL FSUB(.ONE$FLOAT,.TEMP1,.COSINE);
/*   TO COMPUTE THE VALUE OF THE SINE.    */
   CALL FSQR(.COSINE,.TEMP1);
   CALL FSUB(.ONE$FLOAT,.TEMP1,.TEMP1);
   CALL FSQRT(.TEMP1,.SINE);
/*   GIVE SIGNS TO COSINE AND SINE VALUES ACCORDING TO QUADRANT   */
   DO CASE QUAD;
      ;
      ;                                          /*   FIRST QUADRANT   */
      COSINE(3) = COSINE(3) XOR 80H;             /*   SECOND QUADRANT   */
      DO;
         COSINE(3) = COSINE(3) XOR 80H;          /*   THIRD QUADRANT   */
         SINE(3) = SINE(3) XOR 80H;
      END;
      SINE(3) = SINE(3) XOR 80H;                 /*   FOURTH QUADRANT   */
    END;
/*   ALL DONE.  RETURN.    */
  END COS$SIN;
```

```
/*****************************************************************************
*
* ARC$TAN:
*   THIS PROCEDURE IS USED TO CALCULATE THE ARC$TAN FUNCTION IN RADIANS
*   GIVEN AS ARGUMENT A RATIO OF TWO VALUES IN FP REPRESENTATION.
*
* PARAMETERS:
*   - X. - POINTER TO A FOUR BYTE VECTOR REPRESENTING THE DENOMINATOR
*     OF THE RATIO.
*   - Y. - POINTER TO A FOUR BYTE VECTOR REPRESENTING THE NUMERATOR
*     OF THE RATIO.
*   - A. - POINTER TO A FOUR BYTE VECTOR IN WHICH THE VALUE OF THE ANGLE
*     (IN RADIANS) WILL BE PLACED AFTER CALCULATION OF THE ARC$TAN.
*
*****************************************************************************/
ARC$TAN: PROCEDURE (X,Y,A) PUBLIC;
   DCL (X,Y,A) ADDRESS,
       DELTA$X BASED X (4) BYTE,
       DELTA$Y BASED Y (4) BYTE,
       ANGLE BASED A (4) BYTE,
       PI$FLOAT (4) BYTE DATA (0DBH,0FH,49H,40H),        /* 3.141593 */
       TWO$PI (4) BYTE DATA (0DBH,0FH,0C9H,40H),         /* 6.2831853 */
       PI$OVER4 (4) BYTE DATA (0DBH,0FH,49H,3FH),        /* 0.78539819 */
       PI$OVER2 (4) BYTE DATA (0DBH,0FH,0C9H,3FH),       /* 1.5707963 */
       PI$3$OVER2 (4) BYTE DATA (0E4H,0CBH,96H,40H),     /* 4.7123889 */
       CONST$1 (4) BYTE DATA (0F5H,0FFH,7FH,3FH),        /* 0.9999993329 */
       CONST$2 (4) BYTE DATA (1CH,0A6H,0AAH,0BEH),       /* -0.3332985605 */
       CONST$3 (4) BYTE DATA (0A7H,40H,4CH,3EH),         /* 0.1994653599 */
       CONST$4 (4) BYTE DATA (63H,6CH,0EH,0BEH),         /* -0.1390853351 */
       CONST$5 (4) BYTE DATA (0DEH,77H,0C5H,3DH),        /* 0.0964200441 */
       CONST$6 (4) BYTE DATA (0C4H,01H,65H,0BDH),        /* -0.0559098861 */
```

```
        CONST$7 (4) BYTE DATA (51H,16H,0B3H,3CH),          /* 0.0218612288 */
        CONST$8 (4) BYTE DATA (0E6H,0D7H,84H,0BBH),        /* -0.0040540580 */
        CHECK BYTE DATA (04H),
        MSG1(*) BYTE DATA
             (' ARC$TAN FUNCTION UNDEFINED FOR BOTH ARGUMENTS EQUAL TO ZERO.$$'),
        MSG2(*) BYTE DATA (' FATAL ERROR.$$'),
        Z (4) BYTE,
        Z$SQUARE (4) BYTE,
        TEMP (4) BYTE,
        TEMP1 (4) BYTE,
        (SIGN$X, SIGN$Y, ZERO$X, ZERO$Y, I) BYTE;
    SIGN$X, SIGN$Y = 00H;
    DO I = 0 TO 3;
       TEMP(I) = DELTA$Y(I);
       TEMP1(I) = DELTA$X(I);
       END;
/*    SAVE SIGN TO DETERMINE QUADRANT    */
    IF TEMP(3) >= 80H THEN SIGN$Y = 0FFH;
    IF TEMP1(3) >= 80H THEN SIGN$X = 0FFH;
/*    CHECK FOR VALID ARGUMENTS    */
    IF (ZERO$Y := FZTST(.DELTA$Y,.CHECK)) AND
       (ZERO$X := FZTST(.DELTA$X,.CHECK))
    THEN DO;
         CALL CRT$PRINT$STRING (.MSG1);
         CALL CRT$PRINT$STRING (.MSG2);
         HALT;
         END;
    IF ZERO$X
    THEN DO;
         IF SIGN$Y
         THEN DO;
```

```
            DO I = 0 TO 3;
               ANGLE(I) = PI$3$OVER2(I);
               END;
            RETURN;
            END;
         ELSE DO;
            DO I = 0 TO 3;
               ANGLE(I) = PI$OVER2(I);
               END;
            RETURN;
            END;
         END;
/*   FORM Z TO PERFORM HASTINGS APPROXIMATION   */
   TEMP(3) = TEMP(3) AND 7FH;
   TEMP1(3) = TEMP1(3) AND 7FH;
   CALL FSUB(.TEMP, .TEMP1, .Z);
   CALL FADD(.TEMP, .TEMP1, .TEMP);
   CALL FDIV(.Z, .TEMP, .Z);
   CALL FSQR(.Z, .Z$SQUARE);
/*   PERFORM HASTINGS APPROXIMATION FOR ARC$TAN   */
   CALL FMUL(.Z$SQUARE, .CONST$8, .TEMP);
   CALL FADD(.TEMP, .CONST$7, .TEMP);
   CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
   CALL FADD(.TEMP, .CONST$6, .TEMP);
   CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
   CALL FADD(.TEMP, .CONST$5, .TEMP);
   CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
   CALL FADD(.TEMP, .CONST$4, .TEMP);
   CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
   CALL FADD(.TEMP, .CONST$3, .TEMP);
   CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
```

```
    CALL FADD(.TEMP, .CONST$2, .TEMP);
    CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
    CALL FADD(.TEMP, .CONST$1, .TEMP);
     CALL FMUL(.Z, .TEMP, .TEMP);
      CALL FADD(.TEMP, .PI$OVER4, .ANGLE);
 /*    RESTORE ANGLE TO PROPER QUADRANT    */
    IF (NOT SIGN$Y) AND SIGN$X                /* SECOND QUADRANT */
        THEN CALL FSUB(.PI$FLOAT, .ANGLE, .ANGLE);
    IF SIGN$Y AND SIGN$X                     /* THIRD QUADRANT */
       THEN CALL FADD(.PI$FLOAT, .ANGLE, .ANGLE);
    IF SIGN$Y AND (NOT SIGN$X)               /* FOURTH QUADRANT */
       THEN CALL FSUB(.TWO$PI, .ANGLE, .ANGLE);
/*    ALL DONE. RETURN.    */
    END ARC$TAN;


END FLOATING$POINT;
```

```
TIME: DO;


CRT$WRITE:
   PROCEDURE (A) EXTERNAL;
   DECLARE A BYTE; END;

CRT$PRINT$STRING:
   PROCEDURE (A) EXTERNAL;
   DECLARE A ADDRESS; END;

ECHO$CRT:
   PROCEDURE BYTE EXTERNAL;
   END;

SEND$BEL:
   PROCEDURE EXTERNAL;
   END;

SEND$CR:
   PROCEDURE EXTERNAL;
   END;

SEND$CRLF:
   PROCEDURE EXTERNAL;
   END;

SEND$SUB:
   PROCEDURE EXTERNAL;
   END;
```

```
GET$BYTE:
   PROCEDURE (A) BYTE EXTERNAL;
   DECLARE A BYTE; END;

CHECK$INPUT:
   PROCEDURE BYTE EXTERNAL;
   END;

CLEAR$LOW$SCREEN:
   PROCEDURE EXTERNAL;
   END;


DECLARE (MILI$SEC, DUMMY$SEC, SECONDS, MINUTES, HOURS, DAY, SEC$TIME) BYTE PUBLIC,
        TIME$STEP ADDRESS PUBLIC,
        TIME$BUFFER(6) BYTE PUBLIC;
```

```
/*****************************************************************************
*
* CLOCK:
*   THIS PROCEDURE IS OF THE TYPE INTERRUPT.  IT IS USED TO MAINTAIN A REAL
*   TIME HOUR, ONCE INITIATED.  IT USES THE MDS REAL TIME CLOCK.
*
* USAGE:
*   THIS PROCEDURE IS CALLED EACH TIME AN INTERRUPT FROM THE REAL TIME CLOCK
*   IN THE MDS SYSTEM, IS PRODUCED.
*
*                              ***   WARNING   ***
* SINCE THE DEVELOPMENT OF THIS PROCEDURE WAS DONE UNDER ISIS, THE PROCEDURE
* HAD TO BE DECLARED AS INTERRUPT 7, SINCE ISIS DOES NOT ALLOW FOR INTERRUPTS
* LESS THAN OR EQUAL TO 2.  THE INTERRUPT FROM THE REAL TIME CLOCK IS OF LEVEL
* 1, AND THEREFORE, A 'CALL MOVE' INSTRUCTION HAD TO BE IMPLEMENTED, IN
* ORDER TO MOVE THE CODE IN INT 7 TO INT 1 AND PASS OVER THIS ISIS INCONVENIEN-
* CE.  IF A COPY OF THIS PROGRAM IS TO BE EXECUTED WITHOUT ISIS, THEN THIS PRO-
* CEDURE MUST BE RECOMPILED AS PROCEDURE INTERRUPT 1, AND THE 'CALL MOVE' STA-
* TEMENT IN THE INITIATE$CLOCK PROCEDURE, REMOVED.
*
*****************************************************************************/
CLOCK: PROCEDURE INTERRUPT 7;
   DECLARE TEMP BYTE;
/*   TO RESET THE MDS REAL TIME CLOCK.    */
   OUTPUT(0FFH) = 03H;
   MILI$SEC = MILI$SEC + 1;
   IF MILI$SEC = 128 THEN DO;
      MILI$SEC = 0;
      DUMMY$SEC = DUMMY$SEC + 1;
      END;
   IF DUMMY$SEC = 08H THEN DO;
```

```
      MILI$SEC,DUMMY$SEC = 0;
      SEC$TIME = 0FFH;          /*   BOOLEAN VARIABLE.  A SECOND HAS ELAPSED.  */
      SECONDS = SECONDS + 1;
      TIME$STEP = TIME$STEP + 1;
      IF SECONDS = 60 THEN DO;
         SECONDS = 00;
         MINUTES = MINUTES + 1;
         IF MINUTES = 60 THEN DO;
            MINUTES = 00;
            HOURS = HOURS + 1;
            IF HOURS = 24 THEN DO;
               HOURS = 00;
               DAY = 1;
               END;
            END;
         END;
      END;
/*   DISABLE INTERRUPTS.    */
   DISABLE;
/*   RESTORE CURRENT OPERATING LEVEL.    */
   OUTPUT(0FDH) = 020H;
/*   SET THE MDS REAL TIME CLOCK.    */
   TEMP = INPUT(0FFH);
   TEMP = INPUT(0FFH);
   OUTPUT(0FFH) = 00H;
/*   THE RETURN STATEMENT WILL ENABLE INTERRUPTS AUTOMATICALLY.    */
   RETURN;
   END CLOCK;
```

```
/*****************************************************************************
*
* INITIATE$TIME:
*   PROCEDURE USED DURING SYSTEM INITIALIZATION.  USED TO SET THE SIMULATED
*   REAL TIME CLOCK TO AN SPECIFIED HOUR.
*
* USAGE:
*   UTILIZES THE GLOBAL VARIABLES HOURS,MINUTES AND SECONDS.
*
*****************************************************************************/
INITIATE$TIME: PROCEDURE PUBLIC;
   DECLARE OK BYTE;
   DECLARE MSG(*) BYTE DATA (' ***  BAD FORMAT.   ***$$');
   OK = 0;
   DO WHILE OK = 0;
      HOURS, MINUTES, SECONDS = 0FFH;
      CALL CRT$PRINT$STRING(.('INPUT THE TIME AS REQUESTED.$$'));
      CALL SEND$CRLF;
      DO WHILE HOURS >= 24;
         CALL CRT$PRINT$STRING(.('HOURS: $$'));
         HOURS = GET$BYTE(2);
         IF HOURS >= 24
         THEN DO;
            CALL CRT$PRINT$STRING(.MSG);
            CALL SEND$BEL;
            CALL SEND$CR;
            CALL SEND$SUB;
            END;
         ELSE DO;
            CALL CRT$WRITE(17H);        /*  ERASE TO END OF LINE  */
            CALL SEND$CRLF;
```

```
        END;
    END;
DO WHILE MINUTES >= 60;
    CALL CRT$PRINT$STRING(.('MINUTES: $$'));
    MINUTES = GET$BYTE(2);
    IF MINUTES >= 60
    THEN DO;
        CALL CRT$PRINT$STRING(.MSG);
        CALL SEND$BEL;
        CALL SEND$CR;
        CALL SEND$SUB;
        END;
    ELSE DO;
        CALL CRT$WRITE(17H);          /*  ERASE TO END OF LINE  */
        CALL SEND$CRLF;
        END;
    END;
DO WHILE SECONDS >= 60;
    CALL CRT$PRINT$STRING(.('SECONDS: $$'));
    SECONDS = GET$BYTE(2);
    IF SECONDS >= 60
    THEN DO;
        CALL CRT$PRINT$STRING(.MSG);
        CALL SEND$BEL;
        CALL SEND$CR;
        CALL SEND$SUB;
        END;
    ELSE DO;
        CALL CRT$WRITE(17H);          /*  ERASE TO END OF LINE  */
        CALL SEND$CRLF;
        END;
```

```
        END;
    OK = CHECK$INPUT;
    CALL CLEAR$LOW$SCREEN;
    END;

END INITIATE$TIME;
```

```
/*****************************************************************************
*
* INITIATE$CLOCK:
*    PROCEDURE USED TO START THE SIMULATED REAL TIME CLOCK.
*
*****************************************************************************/
INITIATE$CLOCK: PROCEDURE PUBLIC;
   MILI$SEC,DUMMY$SEC = 00;
/*
*                              ***   WARNING   ***
*   THE FOLLOWING STATEMENT SHOULD BE REMOVED IF THIS MODULE IS TO BE
*   EXECUTED WITHOUT ISIS.  THE CLOCK PROCEDURE SHOULD ALSO BE RECOMPI-
*   LED AS PROCEDURE INTERRUPT 1.
*
*/
   CALL MOVE (3,038H,008H);
   OUTPUT(0FDH) = 12H;
   OUTPUT(0FCH) = 00H;
   ENABLE;
   OUTPUT(0FFH) = 00H;
   END INITIATE$CLOCK;
```

```
/*****************************************************************************
*
* ACTUAL$TIME:
*   PROCEDURE USED TO PUT INTO THE VECTOR 'TIME$BUFFER', THE ACTUAL
*   TIME IN A STRING FORM.
*
*****************************************************************************/
ACTUAL$TIME: PROCEDURE PUBLIC;
   TIME$BUFFER(0) = HOURS/10 + 30H;
   TIME$BUFFER(1) = HOURS MOD 10 + 30H;
   TIME$BUFFER(2) = MINUTES/10 + 30H;
   TIME$BUFFER(3) = MINUTES MOD 10 + 30H;
   TIME$BUFFER(4) = SECONDS/10 + 30H;
   TIME$BUFFER(5) = SECONDS MOD 10 + 30H;
   END ACTUAL$TIME;


END TIME;
```

```
PLASMA$PRIMITIVES: DO;



DECLARE LIT LITERALLY 'LITERALLY',
        DCL LIT 'DECLARE';

DCL TRUE LIT '0FFH',
    FALSE LIT '00H';

DCL PLASMA$DATA       LIT '04H',
    PLASMA$STATUS     LIT '05H',
    RECEIVE$MASK      LIT '06H',
    TRANSMIT$MASK     LIT '05H';

DCL STATUS$A          LIT '04H',
    RESET$ALL         LIT '00H',
    OUT$BUSY          LIT '01H',
    IN$BUSY           LIT '02H';

DCL STX               LIT '02H',    /* START TEXT */
    ETX               LIT '03H',    /* ENABLE TEXT */
    CS                LIT '0CH',    /* CLEAR SCREEN */
    CG                LIT '0EH',    /* CONSTRUCT GRAPH */
    CV                LIT '0FH',    /* CLEAR VECTORS */
    EOL               LIT '24H';    /* END OF LINE */
```

PLASMA$PRIMITIVES

```
DCL SET$ERASE          LIT '0100$0000B',
    SET$DASHED         LIT '0001$0000B',
    SET$END            LIT '0010$0000B';
```

```
/*****************************************************************************
*
* SET$STATUS$PLASMA:
*   THIS PROCEDURE IS USED TO SET THE STATUS LINE FOR THE PLASMA.
*   NOTE THAT THE LOGIC TO BE USED IS NEGATIVE.
*
* PARAMETERS:
*   - STATUS. - ASCII CHARACTER USED TO DEFINE THE STATUS LINE.
*
*****************************************************************************/
SET$STATUS$PLASMA: PROCEDURE (STATUS) PUBLIC;
   DCL STATUS BYTE;
   OUTPUT(PLASMA$STATUS) = NOT STATUS;
   END SET$STATUS$PLASMA;
```

```
/*****************************************************************************
*
* PLASMA$WRITE:
*   THIS PROCEDURE IS USED TO SEND A CHARACTER TO THE PLASMA DISPLAY.
*
* PARAMETERS:
*   - CHAR. - ASCII CHARACTER DESIRED TO BE SENT.
*
*****************************************************************************/
PLASMA$WRITE: PROCEDURE (CHAR) PUBLIC;
   DCL CHAR BYTE;
   DO WHILE ((NOT INPUT(PLASMA$STATUS)) AND STATUS$A) <> STATUS$A;
      END;          /* WAIT FOR PLASMA TO BE READY */
   CALL SET$STATUS$PLASMA(RESET$ALL);
   OUTPUT(PLASMA$DATA) = NOT CHAR;
   CALL SET$STATUS$PLASMA(OUT$BUSY);
   END PLASMA$WRITE;
```

```
/*****************************************************************************
*
* CLEAR$PLASMA:
*    THIS PROCEDURE IS USED TO CLEAR THE PLASMA DISPLAY.
*
*****************************************************************************/
CLEAR$PLASMA: PROCEDURE PUBLIC;
    CALL SET$STATUS$PLASMA(IN$BUSY);
    CALL SET$STATUS$PLASMA(RESET$ALL);
    CALL PLASMA$WRITE(CS);
    CALL PLASMA$WRITE(CV);
    CALL SET$STATUS$PLASMA(RESET$ALL);
    END CLEAR$PLASMA;
```

```
/*****************************************************************************
*
* PLASMA$WRITE$VECTOR:
*   THIS PROCEDURE IS USED TO SEND A FOUR BYTE VECTOR TO THE PLASMA.
*
* PARAMETERS:
*   - A. - POINTER TO THE FOUR BYTE VECTOR DESIRED TO BE SENT.
*
*****************************************************************************/
PLASMA$WRITE$VECTOR: PROCEDURE (A) PUBLIC;
   DCL A ADDRESS,
       VECTOR BASED A (4) BYTE,
       VPTR BYTE;
   DO VPTR = 0 TO 3;
      CALL PLASMA$WRITE(VECTOR(VPTR));
      END;
   END PLASMA$WRITE$VECTOR;
```

```
/*****************************************************************************
*
* PLASMA$PRINT$STRING:
*   THIS PROCEDURE IS USED TO WRITE A GIVEN STRING IN A GIVEN POSITION AT
*   THE PLASMA DISPLAY.
*
* PARAMETERS:
*   - COLUMN. - DENOTES THE COLUMN NUMBER DESIRED TO BE ADDRESSED.
*   - ROW. - DENOTES THE ROW NUMBER DESIRED TO BE ADDRESSED.
*   - POINTER. - POINTS TO THE FIRST BYTE OF THE STRING DESIRED TO BE DIS-
*     PLAYED.  NOTE THAT TWO CONSECUTIVE '$' SIGNS MUST MARK THE END OF
*     THE STRING.
*
*****************************************************************************/
PLASMA$PRINT$STRING: PROCEDURE (COLUMN, ROW, POINTER) PUBLIC;
   DCL POINTER ADDRESS,
       BUFFER BASED POINTER (1) BYTE,
       (COLUMN, ROW, COUNT) BYTE;
   COUNT = 0;
   CALL SET$STATUS$PLASMA(IN$BUSY);
   CALL SET$STATUS$PLASMA(RESET$ALL);
   CALL PLASMA$WRITE(STX);
   CALL PLASMA$WRITE(COLUMN);
   CALL PLASMA$WRITE(ROW);
   DO WHILE (BUFFER(COUNT) <> EOL ) OR (BUFFER(COUNT + 1) <> EOL );
      CALL PLASMA$WRITE(BUFFER(COUNT));
      COUNT = COUNT + 1;
      END;
   CALL PLASMA$WRITE(ETX);
   CALL SET$STATUS$PLASMA(RESET$ALL);
   END PLASMA$PRINT$STRING;
```

```
/*****************************************************************************
*
* INITIALIZE$PLASMA:
*   THIS PROCEDURE IS USED TO INITIALIZE THE PLASMA DISPLAY
*
*****************************************************************************/
INITIALIZE$PLASMA: PROCEDURE PUBLIC;
   DCL BUFFER (*) BYTE DATA ('ON LINE.$$');
   CALL CLEAR$PLASMA;
   CALL PLASMA$PRINT$STRING(0, 2, .BUFFER);
   END INITIALIZE$PLASMA;
```

```
/*****************************************************************************
*
* SET$VECTOR:
*   THIS PROCEDURE IS USED PREPARE TWO X AND Y VALUES GIVEN INTO THE FOR-
*   MAT REQUIRED BY THE PLASMA UNIT TO DEFINE A VECTOR.
* PARAMETERS:
*   - X. - ADDRESS VALUE.
*   - Y. - ADDRESS VALUE.
*   - POINTER. - POINTS TO A FOUR BYTE VECTOR IN WHICH THE X AND Y VALUES
*     WILL BE ARRANGED.
*
*****************************************************************************/
SET$VECTOR: PROCEDURE (X, Y, POINTER) PUBLIC;
   DCL (X, Y, POINTER) ADDRESS,
       VECTOR BASED POINTER (4) BYTE;
   VECTOR(0) = CG;
   VECTOR(1), VECTOR(2), VECTOR(3) = 00H;
   VECTOR(1) = LOW(X) AND 07FH;
   VECTOR(2) = LOW(Y) AND 07FH;
   VECTOR(3) = HIGH(SHL(Y AND 180H, 3)) OR
               HIGH(SHL(X AND 180H, 1));
   VECTOR(3) = VECTOR(3) AND NOT SET$ERASE;
   END SET$VECTOR;
```

```
/*****************************************************************************
*
* START$VECTOR$SOLID:
*    THIS PROCEDURE IS USED TO DEFINE A START POINT FOR A SOLID VECTOR.
*
* PARAMETERS:
*    - X. - ADDRESS VALUE.
*    - Y. - ADDRESS VALUE.
*
*****************************************************************************/
START$VECTOR$SOLID: PROCEDURE (X, Y) PUBLIC;
    DCL (X, Y) ADDRESS,
        VECTOR (4) BYTE;
    CALL SET$VECTOR(X, Y, .VECTOR);
    VECTOR(3) = VECTOR(3) AND NOT SET$DASHED;
    VECTOR(3) = VECTOR(3) AND NOT SET$END;
    CALL PLASMA$WRITE$VECTOR(.VECTOR);
    END START$VECTOR$SOLID;
```

```
/*****************************************************************************
*
* STOP$VECTOR$SOLID:
*    THIS PROCEDURE IS USED TO DEFINE A STOP POINT FOR A SOLID VECTOR.
*
* PARAMETERS:
*    - X. - ADDRESS VALUE.
*    - Y. - ADDRESS VALUE.
*
*****************************************************************************/
STOP$VECTOR$SOLID: PROCEDURE (X, Y) PUBLIC;
    DCL (X, Y) ADDRESS,
        VECTOR (4) BYTE;
    CALL SET$VECTOR(X, Y, .VECTOR);
    VECTOR(3) = VECTOR(3) AND NOT SET$DASHED;
    VECTOR(3) = VECTOR(3) OR SET$END;
    CALL PLASMA$WRITE$VECTOR(.VECTOR);
    END STOP$VECTOR$SOLID;
```

```
/*****************************************************************************
*
* START$VECTOR$DASH:
*    THIS PROCEDURE IS USED TO DEFINE A START POINT FOR A DASHED VECTOR.
*
* PARAMETERS:
*    - X. - ADDRESS VALUE.
*    - Y. - ADDRESS VALUE.
*
*****************************************************************************/
START$VECTOR$DASH: PROCEDURE (X, Y) PUBLIC;
    DCL (X, Y) ADDRESS,
        VECTOR (4) BYTE;
    CALL SET$VECTOR(X, Y, .VECTOR);
    VECTOR(3) = VECTOR(3) OR SET$DASHED;
    VECTOR(3) = VECTOR(3) AND NOT SET$END;
    CALL PLASMA$WRITE$VECTOR(.VECTOR);
    END START$VECTOR$DASH;
```

```
/*****************************************************************************
*
* STOP$VECTOR$DASH:
*    THIS PROCEDURE IS USED TO DEFINE A STOP POINT FOR A DASHED VECTOR.
*
* PARAMETERS:
*    - X. - ADDRESS VALUE.
*    - Y. - ADDRESS VALUE.
*
*****************************************************************************/
STOP$VECTOR$DASH: PROCEDURE (X, Y) PUBLIC;
    DCL (X, Y) ADDRESS,
        VECTOR (4) BYTE;
    CALL SET$VECTOR(X, Y, .VECTOR);
    VECTOR(3) = VECTOR(3) OR SET$DASHED;
    VECTOR(3) = VECTOR(3) OR SET$END;
    CALL PLASMA$WRITE$VECTOR(.VECTOR);
    END STOP$VECTOR$DASH;
```

```
/*****************************************************************************
*
* GRAPHIC$DESIG:
*    THIS PROCEDURE IS USED TO DISPLAY THE DESIGNATION OF A CONTACT IN THE
*    NEAREST POSSIBLE ALPHANUMERIC LOCATION TO THE X, Y VALUES GIVEN.
*
* PARAMETERS:
*    - X. - ADDRESS VALUE.
*    - Y. - ADDRESS VALUE.
*    - DESIG. - POINTER TO THE DESIG OF A CONTACT.
*
*****************************************************************************/
GRAPHIC$DESIG: PROCEDURE (X, Y, DESIG) PUBLIC;
   DCL (X, Y, DESIG) ADDRESS,
       VALUE BASED DESIG ADDRESS,
       BUFFER (6) BYTE,
       (ROW, COLUMN) BYTE;
   BUFFER(0) = '[';
   BUFFER(1) = VALUE / 100;
   BUFFER(2) = VALUE MOD 100;
   BUFFER(3) = ']';
   BUFFER(4), BUFFER(5) = EOL;
   ROW = Y / 16;
   COLUMN = ((X - 14) / 6) - 5;
   IF X <= 8 THEN COLUMN = 0;
   IF (X >= 9) AND (X <= 44) THEN COLUMN = X / 6;
   IF (X >= 494) THEN COLUMN = 75;
   CALL PLASMA$PRINT$STRING(COLUMN, ROW, .BUFFER);
   END  GRAPHIC$DESIG;
```

```
END PLASMA$PRIMITIVES;
```

BASICS

```
BASICS: DO;

DECLARE LIT LITERALLY 'LITERALLY',
        DCL LIT 'DECLARE';
DCL CR LIT '0DH',                    /* CARRIAGE RETURN. */
    LF LIT '0AH',                    /* LINE FEED. */
    EOL LIT '24H',                   /* $$ INDICATES END OF LINE. */
    BS LIT '08H',                    /* BACKSPACE.  */
    SPACE LIT '20H',                 /* SPACE */
    RUB LIT '7FH',                   /* RUB OUT.  */
    SUB LIT '1AH',                   /* UP ROW CURSOR.  */
    BEL LIT '07H';                   /* RING THE BELL.  */
DCL TRUE LIT '0FFH',
    FALSE LIT '00H',
    FOREVER LIT 'WHILE TRUE';
DCL CRTDATA LIT '0F6H',              /* CRT DATA ON PORT 246. */
    CRTSTATUS LIT '0F7H',            /* CRT STATUS ON PORT 247. */
    CRT$REC$MASK LIT '06H',          /* MASK FOR CRT: RECEIVE. */
    CRT$TRT$MASK LIT '05H';          /* MASK FOR CRT: TRANSMIT.  */
```

```
/*****************************************************************************
*
* CRT$WRITE:
*   PROCEDURE USED TO SEND A CHARACTER TO THE CRT.
*
* PARAMETERS:
*   - CHAR. -CHARACTER BYTE TO BE SENT.
*
*****************************************************************************/
CRT$WRITE: PROCEDURE (CHAR) PUBLIC;
   DCL CHAR BYTE;
   DO WHILE (INPUT(CRTSTATUS) AND CRT$TRT$MASK) <> CRT$TRT$MASK;   /*WAIT*/
      END;
   OUTPUT(CRTDATA) = CHAR;
   END CRT$WRITE;
```

```
/*****************************************************************************
*
* CRT$PRINT$STRING:
*   PROCEDURE USED TO SEND A STRING OF CHARACTERS TO THE CRT.
*
* PARAMETERS:
*   - A.-POINTER TO STRING.  $$ WILL INDICATE END OF STRING.  MAXIMUM
*        LENGTH: 80 CHARACTERS.
*
*****************************************************************************/
CRT$PRINT$STRING: PROCEDURE (A) PUBLIC;
   DCL (POINTER,A) ADDRESS,
       (BUFFER BASED A) (80) BYTE;
   POINTER = 0;
   DO WHILE (BUFFER(POINTER) <> EOL) OR (BUFFER(POINTER + 1) <> EOL);
      CALL CRT$WRITE(BUFFER(POINTER));
      POINTER = POINTER + 1;
      END;
   END CRT$PRINT$STRING;
```

```
/*****************************************************************************
*
* CRT$READ:
*   PROCEDURE USED TO RECEIVE A CHARACTER FROM THE CRT, AND RETURN ITS
*   VALUE TO THE CALLING MODULE.
*
* USAGE:
*   UNTYPED PROCEDURE.  RETURNS A BYTE VALUE.  (ASCII CODE)
*
*****************************************************************************/
CRT$READ: PROCEDURE BYTE PUBLIC;
   DCL CHAR BYTE;
   DO WHILE (INPUT(CRTSTATUS) AND CRT$REC$MASK) <> CRT$REC$MASK; /*WAIT*/
      END;
   CHAR = INPUT(CRTDATA);
   IF CHAR >= 80H THEN CHAR = CHAR XOR 80H;
   RETURN CHAR;
   END CRT$READ;
```

```
/*****************************************************************************
*
* CRT$TRY$READ:
*   PROCEDURE USED TO TEST IF CHARACTER HAS BEEN SENT FROM CRT.
*
* USAGE:
*   TYPED PROCEDURE.  IF READING WAS SUCCESSFUL, ( A KEY WAS DEPRESSED)
*   THEN THE BYTE VALUE OF THE ASCII CHARACTER WILL BE RETURNED.  NOTICE
*   THAT NO CHARACTER WILL BE DISPLAYED.
*   IF READING WAS NOT SUCCESSFUL, THEN A NUL ASCII CHARACTER (00H) IS
*   RETURNED, AND NO CHARACTER IS DISPLAYED.
*
*****************************************************************************/
CRT$TRY$READ: PROCEDURE BYTE PUBLIC;
   DCL CHAR BYTE;
   CHAR = 00H;
   IF (INPUT(CRTSTATUS) AND CRT$REC$MASK) = CRT$REC$MASK
      THEN CHAR = INPUT(CRTDATA);
   IF CHAR >= 80H THEN CHAR = CHAR XOR 80H;
   RETURN CHAR;
   END CRT$TRY$READ;
```

```
/*****************************************************************************
*
* ECHO$CRT:
*   PROCEDURE USED TO RECEIVE AN ASCII CHARACTER FROM THE CRT, DISPLAY
*   THE SAME, AND RETURN ITS BYTE VALUE TO THE CALLING MODULE.
*
* USAGE:
*   UNTYPED PROCEDURE.  RETURNS A BYTE VALUE.  (ASCII CODE.)
*
*****************************************************************************/
ECHO$CRT: PROCEDURE BYTE PUBLIC;
   DCL CHAR BYTE;
   CHAR = CRT$READ;
   CALL CRT$WRITE(CHAR);
   RETURN CHAR;
   END ECHO$CRT;
```

```
/*****************************************************************************
*
* SEND$SUB:
*   PROCEDURE USED TO SEND AN UP ROW CURSOR CHARACTER TO THE CRT.
*
*****************************************************************************/
SEND$SUB: PROCEDURE PUBLIC;
   CALL CRT$WRITE(SUB);
   END SEND$SUB;
```

```
/*****************************************************************************
*
* SEND$CR:
*   PROCEDURE USED TO SEND A CR ASCII CHARACTER TO THE CRT.
*
*****************************************************************************/
SEND$CR: PROCEDURE PUBLIC;
   CALL CRT$WRITE(CR);
   END SEND$CR;
```

```
/*****************************************************************************
*
* SEND$LF:
*   PROCEDURE USED TO SEND A LF ASCII CHARACTER TO THE CRT.
*
*****************************************************************************/
SEND$LF: PROCEDURE PUBLIC;
   CALL CRT$WRITE(LF);
   END SEND$LF;
```

```
/*****************************************************************************
*
* SEND$CRLF:
*   PROCEDURE USED TO SEND BOTH CR AND LF ASCII CHARACTERS TO CRT.
*
*****************************************************************************/
SEND$CRLF: PROCEDURE PUBLIC;
   CALL CRT$WRITE(CR);
   CALL CRT$WRITE(LF);
   END SEND$CRLF;
```

```
/*****************************************************************************
*
* SEND$BEL:
*   PROCEDURE USED TO SEND A 'BELL' ASCII CHARACTER TO THE CRT.
*
*****************************************************************************/
SEND$BEL: PROCEDURE PUBLIC;
   CALL CRT$WRITE(BEL);
   END SEND$BEL;
```

```
/*****************************************************************************
*
* SEND$BS:
*    PROCEDURE USED TO SEND A NON-DESTRUCTIVE BACKSPACE CHARACTER TO
*    THE CRT.
*
*****************************************************************************/
SEND$BS: PROCEDURE PUBLIC;
    CALL CRT$WRITE(BS);
    END SEND$BS;
```

```
/*****************************************************************************
*
* SEND$SPACE:
*    THIS PROCEDURE IS USED TO SEND SPACES TO THE CRT.
*
* PARAMETERS:
*    - NUM. - NUMBER OF SPACES DESIRED.
*
*****************************************************************************/
SEND$SPACE: PROCEDURE (NUM) PUBLIC;
   DCL NUM BYTE;
   DO WHILE NUM > 0;
      CALL CRT$WRITE(SPACE);
      NUM = NUM - 1;
      END;
   END SEND$SPACE;
```

```
/*****************************************************************
*
* BYTE$CHAR:
*    PROCEDURE USED TO DISPLAY THE DECIMAL VALUE OF A BYTE VARIA-
*    BLE.
*
* PARAMETERS:
*    - CHAR. - BYTE VALUE DESIRED TO BE DISPLAYED.
*
*****************************************************************/
BYTE$CHAR: PROCEDURE (CHAR) PUBLIC;
   DCL VALUE(3) BYTE DATA (100,10,1);
   DCL (I,CHAR,COUNT,TEMP) BYTE;
   DO I = 0 TO LAST(VALUE);
      COUNT = 0;
      TEMP = VALUE(I);
      DO WHILE CHAR >= TEMP;
         CHAR = CHAR - TEMP;
         COUNT = COUNT + 1;
         END;
      CALL CRT$WRITE(COUNT + '0');
      END;
   END BYTE$CHAR;
```

```
/*****************************************************************
*
* ADDRESS$CHAR:
*   PROCEDURE USED TO DISPLAY THE DECIMAL VALUE OF AN ADDRESS
*   VARIABLE.
*
* PARAMETERS:
*   - CHAR. - ADDRESS VALUE DESIRED TO BE DISPLAYED.
*
*****************************************************************/
ADDRESS$CHAR: PROCEDURE (CHAR) PUBLIC;
   DCL VALUE(5) ADDRESS DATA (10000,1000,100,10,1);
   DCL (I,COUNT) BYTE,
       (CHAR,TEMP) ADDRESS;
   DO I = 0 TO LAST(VALUE);
      COUNT = 0;
      TEMP = VALUE(I);
      DO WHILE CHAR >= TEMP;
         CHAR = CHAR - TEMP;
         COUNT = COUNT + 1;
         END;
      CALL CRT$WRITE(COUNT + '0');
      END;
   END ADDRESS$CHAR;
```

```
/*****************************************************************************
*
* BYTE$TO$ASCII:
*   THIS PROCEDURE IS USED TO CONVERT A BYTE QUANTITY INTO TWO ASCII
*   CHARACTERS REPRESENTING ITS HEXADECIMAL VALUE.
*
* PARAMETERS:
*   - A. - POINTER TO THE BYTE QUANTITY DESIRED TO BE CONVERTED.
*   - B,C. - POINTERS TO TWO BYTE QUANTITIES IN WHICH THE TWO ASCII
*            CHARACTERS WILL BE PLACED.  B POINTS TO THE MOST SIGNIFICANT
*            PORTION OF THE ANSWER.
*
*****************************************************************************/
BYTE$TO$ASCII: PROCEDURE (A,B,C) PUBLIC;
   DCL (A,B,C) ADDRESS,
       ENTRY BASED A BYTE,
       OUT1 BASED B BYTE,
       OUT2 BASED C BYTE,
       TEMP BYTE;
   TEMP = SHR(ENTRY,4);
   IF TEMP <= 9 THEN OUT1 = TEMP + 30H;
                ELSE OUT1 = TEMP + 37H;
   TEMP = ENTRY AND 00FH;
   IF TEMP <= 9 THEN OUT2 = TEMP + 30H;
                ELSE OUT2 = TEMP + 37H;
   END BYTE$TO$ASCII;
```

```
/*****************************************************************************
*
* GET$BYTE:
*   PROCEDURE USED TO OBTAIN A DECIMAL NUMBER BETWEEN 0 AND 255 FROM
*   THE CRT.
*
* PARAMETERS:
*   - DIGITS. - INDICATES THE NUMBER OF DIGITS DESIRED. MUST BE LESS THAN
*     OR EQUAL TO 3, ALTHOUGH NO CHECK OF THIS IS MADE.
*
* USAGE:
*   TYPED PROCEDURE THAT WILL RETURN A DECIMAL DIGIT OBTAINED FROM
*   THE CRT AND REPRESENTABLE IN 3 OR LESS DIGITS.
*   THE VALUE RETURNED WILL ALWAYS BE LESS THAN 255.
*
*****************************************************************************/
GET$BYTE: PROCEDURE(DIGITS) BYTE PUBLIC;
   DCL (NUMBER, DIGITS, CHAR, COUNT) BYTE;
   NUMBER, COUNT = 0;
   DO WHILE DIGITS > 0;
      CHAR = CRT$READ;
      DO WHILE (((CHAR < '0') OR (CHAR > '9')) AND (CHAR <> RUB)) OR
      ((CHAR = RUB) AND (COUNT = 0));
         CALL SEND$BEL;
         CHAR = CRT$READ;
         END;
      IF CHAR <> RUB
      THEN DO;
         CALL CRT$WRITE (CHAR);
         NUMBER = NUMBER*10 + (CHAR - 30H);
         COUNT = COUNT + 1;
```

```
      DIGITS = DIGITS - 1;
      END;
   ELSE DO;
      NUMBER = NUMBER/10;
      CALL SEND$BS;
      COUNT = COUNT - 1;
      DIGITS = DIGITS + 1;
      END;
   END;
RETURN NUMBER;
END GET$BYTE;
```

```
/*****************************************************************************
*
* GET$ADDRESS:
*   PROCEDURE USED TO OBTAIN A DECIMAL NUMBER BETWEEN 0 AND 65535 FROM
*   THE CRT.
*
* PARAMETERS:
*   - DIGITS.- INDICATES THE NUMBER OF DIGITS DESIRED.  MUST BE LESS THAN
*     OR EQUAL TO 5, ALTHOUGH NO CHECK OF THIS IS MADE.
*
* USAGE:
*   TYPED PROCEDURE THAT WILL RETURN A DECIMAL VALUE OBTAINED FROM THE
*   CRT AND REPRESENTABLE IN 5 OR LESS DIGITS.  THE VALUE RETURNED WILL
*   ALWAYS BE LESS THAN 65536.
*
*****************************************************************************/
GET$ADDRESS: PROCEDURE(DIGITS) ADDRESS PUBLIC;
   DCL (CHAR,DIGITS,COUNT) BYTE,
       NUMBER ADDRESS;
   NUMBER, COUNT = 0;
   DO WHILE DIGITS > 0;
      CHAR = CRT$READ;
      DO WHILE (((CHAR < '0') OR (CHAR > '9')) AND (CHAR <> RUB)) OR
      ((CHAR = RUB) AND (COUNT = 0));
         CALL SEND$BEL;
         CHAR = CRT$READ;
         END;
      IF CHAR <> RUB
      THEN DO;
         CALL CRT$WRITE(CHAR);
         NUMBER = NUMBER*10 + (CHAR - 30H);
```

```
        COUNT = COUNT + 1;
        DIGITS = DIGITS - 1;
        END;
     ELSE DO;
        CALL SEND$BS;
        NUMBER = NUMBER/10;
        COUNT = COUNT - 1;
        DIGITS = DIGITS + 1;
        END;
     END;
  RETURN NUMBER;
  END GET$ADDRESS;
```

```
/*****************************************************************************
*
* GET$STRING:
*   PROCEDURE USED TO OBTAIN A STRING OF 'NUMBER' CHARACTERS FROM THE
*   CRT.
*
* PARAMETERS:
*   - NUMBER. - NUMBER OF CHARACTERS DESIRED.
*   - A. - POINTER TO A MEMORY LOCATION IN WHICH THE STRING IS DESIRED
*     TO BE PLACED.
*
* USAGE:
*   UNTYPED PROCEDURE.  THE ONLY CHARACTERS THAT CAN BE ACCEPTED FROM
*   THE CRT ARE: NUMBERS, UPPER AND LOWER CASE ALPHABETICS, ALTHOUGH
*   ALL LOWER CASE ALPHABETICS WILL BE CONVERTED INTO UPPER CASE.
*
*****************************************************************************/
GET$STRING: PROCEDURE(A,NUMBER) PUBLIC;
  DCL A ADDRESS,
      (CHAR BASED A, TEMP, NUMBER,COUNT) BYTE;
   COUNT = 0;
   DO WHILE NUMBER > 0;
      CHAR = CRT$READ;
      DO WHILE (((CHAR < 30H) OR (CHAR > 7AH) OR ((CHAR > 39H) AND (CHAR < 41H))
      OR ((CHAR > 5AH) AND (CHAR < 61H))) AND (CHAR <> RUB)) OR
      ((CHAR = RUB) AND (COUNT = 0));
         CALL SEND$BEL;
         CHAR = CRT$READ;
         END;
      IF (CHAR >= 61H) AND (CHAR <= 7AH)
         THEN CHAR = CHAR - 20H;
```

```
      IF CHAR <> RUB
      THEN DO;
         CALL CRT$WRITE (CHAR);
         A = A + 1;
         NUMBER = NUMBER - 1;
         COUNT = COUNT + 1;
         END;
      ELSE DO;
         CALL SEND$BS;
         A = A - 1;
         NUMBER = NUMBER + 1;
         COUNT = COUNT - 1;
         END;
   END;
   END GET$STRING;
```

```
/*****************************************************************************
*
* PUT$NUMBER$BUFFER:
*   THIS PROCEDURE IS USED TO OBTAIN AN SPECIFIED NUMBER OF NUMERIC CHARACTERS
*   FROM THE CRT, AND TO PUT THEM IN A GIVEN MEMORY LOCATION.
*
* PARAMETERS:
*   - NUM. - NUMBER OF NUMERIC CHARACTERS DESIRED.
*   - A. - POINTER TO A MEMORY LOCATION IN WHICH THE STRING IS DESIRED TO
*          BE PLACED.
*
*****************************************************************************/
PUT$NUMBER$BUFFER: PROCEDURE(NUM,A) PUBLIC;
   DCL A ADDRESS,
       (TEMP BASED A, NUM, CHAR, COUNT) BYTE;
   COUNT = 0;
   DO WHILE NUM > 0;
      CHAR = CRT$READ;
      DO WHILE (((CHAR < '0') OR (CHAR > '9')) AND (CHAR <> RUB)) OR
      ((CHAR = RUB) AND (COUNT = 0));
         CALL SEND$BEL;
         CHAR = CRT$READ;
         END;
      IF CHAR <> RUB
      THEN DO;
         CALL CRT$WRITE(CHAR);
         TEMP = CHAR;
         A = A + 1;
         NUM = NUM - 1;
         COUNT = COUNT + 1;
         END;
```

```
      ELSE DO;
         CALL SEND$BS;
         A = A - 1;
         NUM = NUM + 1;
         COUNT = COUNT - 1;
         END;
      END;
   END PUT$NUMBER$BUFFER;


END BASICS;
```

LIST OF REFERENCES

1. Babin, O. P. and Seaman, R. S., A Microcomputer Based Plasma Display System, M. S. Thesis Naval Postgraduate School, Monterey, California, 1978.

2. Elite 2500 Instruction Manual, Datamedia Corporation, 7300 N. Crescent Blvd., Pennsauken, N. J., 08110.

3. Kerns, K. H. and Cooper, R. S., A Microcomputer Solution to Maneuvering Board Problems, M.S. Thesis Naval Postgraduate School, Monterey, California, 1973.

4. Hastings, C.Jr., Approximations for Digital Computers, Princeton, New Jersey, Princeton University Press, 1955.

5. Intellec 800 Microcomputer Development System Operator's Manual, Intel Corporation, Santa Clara, California, 1975.

6. Intellec Microcomputer Development System Hardware Reference Manual, Intel Corporation, Santa Clara, California, 1976.

7. ISIS-II PL/M 80 Compiler Operator's Manual, Intel Corporation, Santa Clara, California, 1976.

8. ISIS-II System User's Guide, Intel Corporation, Santa Clara, California, 1976.

9. PL/M 80 Programming Manual, Intel Corporation, Santa Clara, California, 1976.

10. SBC 310 High-Speed Mathematics Unit Hardware Reference Manual, Intel Corporation, Santa Clara, California, 1977.

11. Newman, W. M. and Sproull, R. F., Principles of Interactive Computer Graphics, McGraw-Hill Computer Science Series, 1973.

12. Operation Specialist 3 & 2 (UNCLASSIFIED), Naval Education and Training Command, Rate Training Manual and Nonresident Career Course, NAVEDTRA 10144-C, 1975.

13. Plasma Display Set Technical Manual Vol. I, Science Applications Inc., San Diego, California, 1976.

14. Plasma Display Set Technical Manual Vol. II, Science Applications Inc., San Diego, California, 1976.

15. Scheid, F.,Theory and Problems of Numerical Analysis, Schaum's Outline Series, McGraw Hill, 1968.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Fleet Support Office Naval Ocean Systems Center San Diego, California 92152	1
3.	Dr. William A. Von Winkle Associate Technical Director for Technology Naval Underwater Systems Center New London, Connecticut 06320	1
4.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
5.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	3
6.	LCDR Stephen T. Holl, USN, Code 52Hl Department of Computer Science Naval Postgraduate School Monterey, California 93940	3
7.	Professor George A. Rahe, Code 52Ra Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
8.	Associate Professor Uno R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93940	3

9. LCDR Antonio Luiz S. Goncalves, Brazilian Navy 6
Rua Prudente de Moraes, 660 apto. 202
Ipanema, Rio de Janeiro 20000
RJ, BRAZIL

10. LT(JG) Javier E. De la Cuba Bravo, Peruvian Navy 6
Direccion de Instruccion de la Marina
Ministerio de Marina
Ave. Salaverry S/N
Lima, PERU

11. LCDR Ken Clare, USN 1
Code 112 NAVDAC
Washington, D.C. 20374

12. Dr. Joel Trimble 1
Office of Naval Research
Arlington, Virginia 22217

13. LT Mark S. Moranville, USN, Code 52Mi 1
Department of Computer Science
Naval Postgraduate School
Monterey, California 93940

Thesis
G5469
pt.1
c.1

177100

Goncalves
A microcomputer based shipboard surface-sub-surface contact plotter system.

4 OCT 78 24924
30 JAN 79 24428
19 AUG 83 27914
29 NOV 84 27986

Thesis
G5469
pt.1
c.1

177100

Goncalves
A microcomputer based shipboard surface-sub-surface contact plotter system.

thesG5469pt.2
A microcomputer based shipboard surface-
3 2768 002 13086 6
DUDLEY KNOX LIBRARY